An Overview of Hardware Implementation of Membrane Computing Models

GEXIANG ZHANG, Chengdu University of Technology, China and Southwest Jiaotong University, China
ZEYI SHANG, Southwest Jiaotong University, China and Université Paris-Est Créteil Val de Marne, France
SERGEY VERLAN, Université Paris-Est Créteil Val de Marne, France
MIGUEL Á. MARTÍNEZ-DEL-AMOR, Universidad de Sevilla, Spain
CHENGXUN YUAN, Southwest Jiaotong University, China
LUIS VALENCIA-CABRERA and MARIO J. PÉREZ-JIMÉNEZ, Universidad de Sevilla, Spain

The model of membrane computing, also known under the name of P systems, is a bio-inspired large-scale parallel computing paradigm having a good potential for the design of massively parallel algorithms. For its implementation it is very natural to choose hardware platforms that have important inherent parallelism, such as field-programmable gate arrays (FPGAs) or compute unified device architecture (CUDA)-enabled graphic processing units (GPUs). This article performs an overview of all existing approaches of hardware implementation in the area of P systems. The quantitative and qualitative attributes of FPGA-based implementations and CUDA-enabled GPU-based simulations are compared to evaluate the two methodologies.

The work of G.Z., Z. S., and S. V. is supported by the National Natural Science Foundation of China (61972324, 61672437, 61702428), Beijing Advanced Innovation Center for Intelligent Robots and Systems (2019IRS14), Artificial Intelligence Key Laboratory of Sichuan Province (2019RYJ06), the New Generation Artificial Intelligence Science and Technology Major Project of Sichuan Province (2018GZDZX0043), and the Sichuan Science and Technology Program (2018GZ0086). M.A.M-d-A., L.V-C., and M.J.P-J. also acknowledge the support of the research project TIN2017-89842-P (MABICAP), co-financed by *Ministerio de Economía, Industria y Competitividad (MINECO)* of Spain, through the *Agencia Estatal de Investigación (AEI)*, and by *Fondo Europeo de Desarrollo Regional (FEDER)* of the European Union.
Authors' addresses: G. Zhang (corresponding author), Chengdu University of Technology, No.1, Dongsan Road, Erxianqiao, Chenghua District, Chengdu, 610059, China, Southwest Jiaotong University, 999 Xi'an Rd, Chengdu, 611756, China; email: zhgxdylan@126.com; Z. Shang, Southwest Jiaotong University, 999 Xi'an Rd, Chengdu, 611756, China, Université Paris-Est Créteil Val de Marne, 61 av. du général de Gaulle, Créteil, 94010, France; email: zeyi.shang@lacl.fr; S. Verlan, Université Paris-Est Créteil Val de Marne, 61 av. du général de Gaulle, Créteil, 94010, France; email: verlan@u-pec.fr; M. Á. Martínez-del-Amor, L. Valencia-Cabrera, and M. J. Pérez-Jiménez, Universidad de Sevilla, Avda. Reina Mercedes S/N, Seville, 41012, Spain; emails: {mdelamor, lvalencia, marper}@us.es; C. Yuan, Southwest Jiaotong University, 999 Xián Rd, Chengdu, 611756, China; email: 502220232@qq.com.

1 INTRODUCTION

With the arising of *protocells* 3.84B years ago in hydrothermal vent precipitates [26], the evolution of unicellular organisms led to the emergence of multicellularity [83]. The biological cell structure defined by the membranes has evolved and was optimized for billions of years. As a consequence, a biological cell is a powerful parallel processing unit that can perform sophisticated biologic behaviors. Inspired by the structure of biological membranes and by internal biochemical reactions, Gheorghe Păun initiated the area of membrane computing in 1998 as a theoretical com-puter science model borrowing many concepts from the cell biology [84]. The model, also called P systems, features a nested membrane-like structure delimiting regions that host objects (mod-eling cell chemicals) that are transformed locally or communicated to other regions by different types of rules that mimic cell chemical transformations. The evolution of the state of the system is performed by transition steps doing a synchronous parallel execution of all rules. Since a typical model contains hundreds and even thousands of such parallel executions, P systems feature an inherent massive parallelism and the global behavior of the system is emerging from many simple interactions provided by rules application. There exist numerous variants of the model depend-ing on the manipulated objects, used types of rules, and the parallel execution strategy (called derivation mode).

Membrane Computing is a computing paradigm inspired from the structure and functioning of living cells, and the organization of cells in tissues and other structures, including the brain. It provides distributed parallel computing devices called, generically, P systems. Membrane computing models are an extension of DNA computing, a possible source of novel/useful/interesting computing models. If one wants to model (1) discrete, (2) distributed, (3) parallel, (4) cell-like or tissue-like systems, (5) dealing with multisets, (6) evolving by rewriting-like rules, then P systems are obligatory/unavoidable and the unique models dealing with all these features [85]. As unconventional computing devices within natural computing, P systems have proved to overcome the well-known limitations imposed by the conventional techniques based on electronic technology. More specifically, the relevance of these computing models can be non-exhaustively summarized in the following points. From a theoretical view, a novel methodology to tackle the famous **P** versus **NP** problem has been given [76]. The main focus in the area was the introduction of (biologically inspired) variants of P systems and the further study of their computational power, in particular of their computational completeness (see Reference [87] summarizing hundreds of articles on this topic). From a practical view, the theoretical investigation led to a series of successful applications in different areas ranging from image processing [25, 114], meta-heuristic algorithms for optimization problems [116, 117, 119], robot controllers [14, 109], path planning [79, 81, 110], and fault diagnosis for electrical power systems [90, 107, 108]. Due to its historical background, membrane computing was also used as a modeling framework for biological and ecological subjects such as artificial life [94], photosynthesis [75], p53 protein signaling pathway [95], myxobacterial colony [68], and biopolymer duplication [56]. (Please see a recent overview in Reference [118]).

For 20 years many applications of P systems have arisen, not only in Theoretical Computer Science, but also in Computational Modelling, Robotics, Optimization, and so on, as mentioned

above. The interest in this area is increasing, given that this unconventional, massively parallel approach has been demonstrated to provide a powerful, flexible, and expressive framework. One example is the 2016 report of the National Research Council of Canada where Membrane Computing appears to be mentioned several times as prominent parts of bio-computing [112]. Therefore, there is a need for simulators to build model validation tools, assistants for formal verification, environments for virtual experimentation, model calibration tools, and so on. So far, most of the simulation software aims at reproducing only the computation results instead of procedures. However, there has also been a research line concerning the implementation of P system parallelism in real parallel platforms. The main hardware employed for this is FPGAs and GPUs, given their efficiency, fast shared memory system, and scalability. The objective is twofold: on the one hand, provide efficient simulation tools where the massive, natural P system parallelism do not get serialized and is instead harnessed to speed up the simulations; and on the other hand, to explore how this special parallelism can be mapped in current modern computing architectures. It is known that the future of computer architecture will go through non–Von Neumann architectures [12]. Natural Computing models can provide solutions where the instructions are close to, or along with, the data. P systems are just an alternative, like artificial neural networks, for this. We want to shed a light into this alternative from the perspective of efficient implementation, analyzing all the challenges and current solutions. In addition, a number of these applications mentioned above only theoretically profit from the potential speedup promised by the model, as there are no truly parallel implementations available. However, the inherent large-scale parallelism of the model has the profound potential for the progress of extreme data processing, especially taking into account that there are not so many widely investigated massively parallel computational paradigms. Thus, an interesting topic is the implementation of membrane computing models on contemporary silicon integrated circuits. This allows to exploit the desirable parallel computational capability of P systems to explore a new orientation for high performance computing (HPC) [51, 54].

As pointed out in Reference [43] "some unmistakable trends in hardware design indicate that uniprocessor (or implicitly parallel) architectures may not be able to sustain the rate of realizable performance increments in the future." The augmentation of electronic ingredients' density had been subject to the well-known Moore's law for decades. After extraordinary exponential growth of many years, the number of transistors in chips cannot follow this law, at least it cannot be doubled within two years [2]. With transistors shrunk to nanoscale, quantum effects stand out [66] and the behavior of circuits is not up to expectations. Moreover, even with the further increase of the density, the computational capability growth is not linearly proportional to it [111]. Another knotty problem is the heat dissipation, which would melt the silicon substrate with density level increasing. Traditional semiconductor scaling is predicted to reach an end by about 2024 on the foundation of prior arts [1]. Parallel computing has the potential to further uplift computing power provided that the density of transistor is constant [101] with multicore and multithread architecture, although heat dissipation and interconnect issues would be challenges [8].

All these observations give strong arguments in favor of investigation of parallel computing platforms. Before claiming that the era of parallel computing has dawned, one essential question should be clarified: What does parallel computing mean? Though not rigorous, parallel computing implies computing based on the decomposition of the task into a set of concurrently executable operations and the assignment of these operations to multiple parallel processing nodes. The evolution of computer processor scheme, from single-core single central processing unit (CPU) to multiple-core single CPU and multiple-core multiple-CPU frame is an instance of parallel computing. The inherent parallelism of P systems place them in the class of multiple processing nodes computing devices. The mapping of constituents to processing nodes gives rise to different

implementation strategies. A common practice in the area of natural computing is to observe natural processes and somehow mimic the corresponding behavior. The way living cells allocate, organize, and coordinate processing nodes has evolved for billions of years. Investigating this magnificent course might help us to handle the multiple cores computing, which has cut a striking figure in the contemporary parallel computing realm.

The large-scale distributed parallel processes occurring in vesicle compartments and the vesicle division functionality enlightened from mitosis of living cells are two of the most outstanding advantages of membrane computing. They allow to underlie the foundation for the construction of highly parallel computation platform whose performance, flexibility, and scalability outperforms traditional sequential counterparts substantially [118]. As a parallel computing paradigm inspired by the structural and functional features of biological membranes, only parallel computing platforms are suitable for the implementation of P systems. More precisely, the limited parallelism of general computers realized by the communication mechanism among the multiple cores of CPU and GPU cannot make full use of the large-scale parallelism, non-determinism, and other particular attributes that impart an enormous computing potential such as creation and dissolution of inner membranes, the self-replication or *autopoiesis* [29] of the whole cell-like entirety that works as a computing unit, the communication by *symport* and *antiport* of objects [4], and so on. We would like to remark that programming membrane computing algorithms with high-level general purpose languages and executing them on the computer represents just a simulation and not a real implementation of P systems [85].

Several software-based and hardware-based parallel computing platforms have been developed to implement P systems. The first software-based parallel computing platform was constructed using a cluster of computers [22]. This platform achieves good performance and flexibility. Nonetheless, when the size of target P system is increased, the consumption of CPU time and resources caused by the communication between different computers rises dramatically. Moreover, the underlying hardware (a cluster of computers) of this platform cannot be miniaturized closing the way to corresponding membrane computing algorithms to be used in embedded chips and compact controllers, which can be employed in robots, automobiles, machine tools, and so on. This disadvantage limits the range of applications of such platforms for membrane computing.

Hence, it is important to propose hardware implementations of P systems as specific architectures that do not have the drawbacks related to the traditional ways of implementation. There are two main directions for such research using (1) field-programmable gate arrays (FPGA) and (2) graphical processing units (GPUs) relying on compute unified device architecture (CUDA) platform. In the first case a completely new parallel circuit is specially designed to implement some variants of P systems. In the second case the pre-defined CUDA parallel platform is used to simulate P systems. The achieved performance and correspondence is smaller in this case, but the development effort is much lower, so finally it becomes an interesting compromise between traditional computers implementations and a highly parallel one using specialized circuits. Also, when implementing membrane computing models in hardware, the main difficulty comes from the fact that there exists a big number of variations of the basic model of P systems having quite distinct characteristics [88]. This poses a great challenge for the conception of a general computational architecture to implement these various models.

The computation in a P system is a sequence of transitions between configurations. The core problem for implementations is the *object distribution problem* (ODP) that computes one of the multisets of applicable rules to the current configuration and whose application permits to reach the next configuration. This problem is a particular variant of a more general problem that computes the whole applicable set of multisets of rules for a given configuration and it is known to be NP-complete [21]. Known algorithms and heuristics do not parallelize well, so special

heuristics were developed to quickly compute the desired multiset of rules. We decided to present these heuristics uniformly in terms of multi-criteria optimization (see Sections 4 and 5). Another problem for implementations is that in the general case the model is non-deterministic, so an equitable choice among different possibilities should be provided. However, this is very difficult to achieve and in most of the cases this property is not satisfied.

This article is organized as follows: Section 2 gives a brief description of the capabilities and of the architecture of hardware used, Section 3 recalls the definition of the most general model of P systems, based on the formal framework [35]. Next, Section 4 expresses object distribution problem (ODP) in terms of multi-criteria optimization and integer linear programming. Section 5 presents an overview of different simulation approaches, including the Direct Non-deterministic Distribution algorithm (DND) algorithm, which is the base for handling non-determinism. Next, Sections 6, 7, and 8 give more details on existing simulation approaches. At the end, conclusions and future research directions are discussed.

2 THE SELECTION OF HARDWARE

2.1 FPGA Hardware

FPGA is a reconfigurable hardware allowing to prototype digital circuits. The modification of circuits for FPGA is performed by altering the interconnections between circuit elements. FPGA is developed by hardware description language, the most common ones being VHSIC Hardware Description Language (VHDL) and Verilog. The design can be performed on several levels of abstraction, ranging from the switch/transistor level until the behavioral level (corresponding to a Mealy machine [67]).

From structural point of view, an FPGA is an array of configurable logic blocks (CLB) inlaid in the matrix of interconnects. CLBs comprise slice-organized logic cells which are arranged in a way named \emph{look-up table} (LUT). LUTs are used to implement different combinatorial circuits such as basic gates, decoders, encoders, and multiplexers. Logic cells contain a type of storage element called flip-flop (FF) which is used as memory elements in sequential circuits. To perform particular operations, interconnects of CLBs should be reconfigured. The interconnection in FPGA is performed by switch boxes routing signals between its logic blocks. Modern FPGAs use one of the following interconnect technologies: static RAM, flash memory, and anti-fuse. The first one dominates the current FPGAs implementations. With the re-programmability, FPGAs comply with the model-oriented hardware implementation quite well.

2.2 CUDA-enabled GPU Hardware

Nowadays, multi-core architecture CPU is the mainstream. However, the component integration scale of some high-end GPUs has outpaced the CPUs for the booming demand of graphics processing (advanced rendering and 3D vision) [52]. Currently, the GPU is a computing element as powerful as the CPU. Different from FPGAs, there are manufactured parallel architectures in GPUs. The advantage is that developers should just be concerned about the efficient utilization of these architectures, and the drawback is that these frameworks are un-reconfigurable.

Nevertheless, the GPU is not a general processing unit that can handle other computing assignments except for graphics processing. This predicament has changed for the arise of *compute unified device architecture*, known as *CUDA*, from leading chip vendor NVIDIA corporation. CUDA is a technology that enables *general-purpose computing on graphics processing units (GPGPU)*. Usually, when referring to CUDA, what is referred is not the parallel computing framework but a GPU supporting CUDA. A CUDA-enabled graphics processing unit is a universal parallel computing device that is suitable for the implementing of parallel algorithm models. The parallel computing

behavior of CUDA is based on the execution of multiple compute *kernels* on the GPU. These compute kernels are without physical construction, but based on an abstract parallel programming model. In other words, CUDA does not alter the physical structure of GPU. The CUDA-enabled GPU does not work alone, but form a heterogeneous computing architecture with the CPU, where the CPU (*host*) is the master node that controls the execution flow and launches kernels on the GPU (*device*) when massive parallelism is required [52]. A kernel is executed by a *grid* of (thousands of) *threads*. The grid is a two-level hierarchy, where *threads* are arranged into *thread blocks* of equal size. Each block and each thread is unequivocally identified by an identifier. In this way, threads and blocks can be distributed easily to different portions of data or to compute different instructions. Threads from the same block can be synchronized using *barriers*, while those belonging to different blocks can only be synchronized by the end of the execution of the kernel.

A GPU contains a *global memory*, which has the biggest size, but has the longest access time and a *shared memory*, which is smaller but faster [52]. Although current GPUs contain cache memories, to accelerate memory accesses, best performance is achieved when doing it manually. Global memory is accessed by all threads launched in all grids, and also by the host, but shared memory is only accessible by threads in the same block. Threads also have fast access to their own registers and local memory (which is normally outsourced to global memory). Accesses to memory have to be carefully programmed, so that contiguous portion of data is read by consecutive threads (providing so-called *coalesced access*), since this increases the memory bandwidth utilization.

Nowadays the architecture of GPUs is upgraded to *Streaming Multiprocessors (SMs)* that are composed of an array of *Streaming Processors (SPs)*, working as computing cores. A thread set consisting of 32 threads named *warp* is the basic unit that an SM fulfills in its executions. An SM can manage multiple warps that are based on *Single-Instruction Multiple-Thread (SIMT)* model, in effect. Each thread in a warp should commence its processing at the identical program address concurrently, although after beginning, threads can execute independently abiding by a sequential manner. The parallelism of CUDA is terminated when a warp branches or the memory stalls [61].

2.3 Other Hardware

There were attempts to simulate certain types of P systems on micro-processor–based architectures [47]. Although the performance of micro-processors is low, they are economical alternatives suitable for the developing of prototypes and verifying the design methods. The drawback is that in most of the cases the parallel nature of P systems is not exploited at all, often leading to inefficient implementations. It could be interesting to use out-of-order (OoO) execution [93] processors to tackle this problem, but this possibility was not investigated yet.

Custom application-specific integrated circuits (ASIC) can also be employed to implement P systems. However, no such attempts exit at the moment of the writing of this article. One of the main reasons is that the flexibility of the hardware platform constructed on ASIC is quite insufficient to adapt to the different variants of P systems. However, such attempts could still be meaningful, as they would allow to simulate features of P systems on some tailored circuits with different properties, like low power consumption. Another possible direction is to use ASIC for particular commonly occurring computational cores and combine them with software execution [28, 103].

3 THE MODEL OF P SYSTEMS

We suppose the reader has a knowledge of basic notions from formal language theory and membrane computing. We refer to References [37, 87, 91] for missing details. We will also closely follow References [35, 105] for the definitions.

We recall that a multiset can be seen as a set whose elements can have greater than one multiplicity. We will use the string notation for multisets, i.e., a multiset M will be represented by a string where the number of occurrences of each letter corresponds to its multiplicity in M. We will denote by $|M|$ the size of the multiset M and by $|M|_a$ the number of elements a in M.

There exist many variants of P systems (see, e.g., Reference [85]). In this article the presentation will be given in terms of the *formal framework for P systems* introduced in Reference [35], see also References [104, 105]. This framework is based on the model of *network of cells* specifying the structure and rules to be executed as well as on a set of four functions giving the semantics of the system. Different combinations of these parameters allow in most of the cases to construct for a given P system a network of cells that strongly bisimulates it (i.e., one step in the one system is simulated by one step in the other one). Moreover, in many cases there is a one-to-one correspondence between rules applied in each system, meaning that they are mostly indistinguishable. Hence, the framework for P systems can be seen as a common language to compare different P systems as well as to express notions related to this area. Moreover, other multiset rewriting-based models (like Petri nets) can be easily expressed into this framework giving a possibility to compare corresponding models. Finally, we would like to remark that in what follows, we will use the variant of the framework supposing that the system structure does not change in time. An extension of the framework that permits to take into account notions related to P systems with dynamically evolving structure is given in Reference [34], but corresponding definitions are too complex for the purpose of this article.

3.1 Network of Cells

As pointed out in References [33, 35, 104, 105], most types of (static structure) P systems can be seen as variants of parallel multiset rewriting (by using the algorithm called *flattening*). Since multiset rewriting level is not practical for system description and understanding, a higher-level concept called *network of cells* was introduced in Reference [35]. This model augments multiset rewriting with the notion of spatial locations (*cells*) as well as the corresponding operations and it can be seen as a particular interpretation of the symbols. References [35, 104, 105] define some basic building blocks in terms of network of cells and give several examples of the construction of widespread notions and types of rules in membrane computing using these blocks.

Below, we provide the definition of *network of cells*, taken from Reference [35]. We remark that the definition from Reference [34] is slightly different, however, both models coincide when the structure of the system does not evolve.

Definition 3.1 ([35]). A *network of cells of degree* $n \geq 1$ is a construct

$$\Pi = (n, V, w, Inf, R),$$

where

(1) n is the number of cells;
(2) V is an *alphabet*;
(3) $w = (w_1, \ldots, w_n)$ where $w_i \in V^\circ$, for all $1 \leq i \leq n$, is the *finite multiset initially associated to cell* i;
(4) $Inf = (Inf_1, \ldots, Inf_n)$, where $Inf_i \subseteq V$, for all $1 \leq i \leq n$, is the *set of symbols occurring infinitely often in cell* i (in most of the cases, only one cell, called the *environment*, will contain symbols occurring with infinite multiplicity);

(5) R is a finite set of rules of the form

$$(X \to Y; P, Q),$$

where $X = (x_1, \ldots, x_n)$, $Y = (y_1, \ldots, y_n)$, $x_i, y_i \in V^\circ$, $1 \le i \le n$, are vectors of multisets over V and $P = (p_1, \ldots, p_n)$, $Q = (q_1, \ldots, q_n)$, p_i, q_i, $1 \le i \le n$ are finite sets of multisets over V. We will also use the notation (omitting p_i, q_i, x_i or y_i if they are empty)

$$(1, x_1) \ldots (n, x_n) \to (1, y_1) \ldots (n, y_n); [(1, p_1) \ldots (1, p_n)]; [(1, q_1) \ldots (n, q_n)].$$

The above rule is applied as follows: objects x_i from cells i are rewritten into objects y_j produced in cells j, $1 \le i, j \le n$, if every cell k, $1 \le k \le n$, contains all multisets from p_k and does not contain any multiset from q_k.

By taking for each rule the set of cells that are involved a hypergraph relation, called the *structure* of the system, is induced. Commonly, tree-like (for P systems) or graph-like (for tissue P systems) relations are considered.

The *configuration* C of Π is defined as an n-tuple of multisets over V $(u_1, \ldots, u_n)$ satisfying $u_i \cap Inf_i = \emptyset$, $1 \le i \le n$.

To define the *computation* in network of cells according to some derivation mode δ the following functions should be specified:

- $Applicable(\Pi, C, \delta)$ – the function taking a system Π, a configuration C, and a derivation mode δ and yielding the set of multisets of rules of Π that can be applied to C.
- $Apply(\Pi, C, R)$ – the function allowing to compute the configuration obtained by the parallel application of the multiset of rules R to the configuration C.
- $Halt(\Pi, C, \delta)$ – a predicate that yields true if C is a halting configuration of the system Π (in some derivation mode δ).
- $Result(\Pi, C)$ – a function giving the result of the computation of the P system Π when the halting configuration C has been reached.

Then the computation is a sequence of transitions where each transition step $C \Rightarrow C'$ is defined as $C' = Apply(\Pi, C, R)$, for some $R \in Applicable(\Pi, C, \delta)$. As usual, this sequence starts with the initial configuration and ends with the final configuration for which the halting predicate $Halt$ yields true. In more formal terms, the result of the computation of a network of cells $\Pi = (n, V, w, Inf, R)$ working in the derivation mode δ is defined as follows (we refer to Reference [35] for more technical details):

$$Result(\Pi) = \{Result(\Pi, z) : w \Rightarrow^* z, Halt(\Pi, z, \delta) = true \text{ and } Halt(\Pi, x, \delta) = false \text{ for any } x : w \Rightarrow^* x \Rightarrow^+ z\}.$$

We remark that Reference [35] gives an algorithm to compute the set of multisets of applicable rules, denoted as $Applicable(\Pi, C, asyn)$, the algorithm to compute $Apply(\Pi, C, R)$, and several definitions for $Halt$ and $Result$ functions. The most common way of halting is the *total* halting, which means that there are no more applicable rules and the most common way of getting the result is to consider the multiset of objects present in the halting configuration at some predefined cell.

At each step, the set of multisets of applicable rules $Applicable(\Pi, C, asyn)$ can be restricted using a *derivation mode*, denoted by δ, which specifies which sub-multisets are chosen for the next step application. The most common example is the maximally parallel derivation mode (max), which is defined as follows:

$$Applicable(\Pi, C, max) = \{R \subseteq Applicable(\Pi, C, asyn) \mid \nexists R' \in Applicable(\Pi, C, asyn) : R' \supsetneq R\}.$$

The filter condition states that only non-extensible multisets are considered in this derivation mode. We remark that there might be several such multisets, hence for the application a non-deterministic choice is used to take one of them. We also refer to Reference [105] for a description of several other derivation modes.

Example 3.2. Consider the system $\Pi = (O, w_1, R)$, having the alphabet $O = \{a, b, c\}$ and the set of rules $R = \{r_1 : (1, ab) \to (1, abc); r_2 : (1, bbc) \to (1, abb)\}$. Consider the configuration $C = (1, a^3b^4c^2)$. Then, $Applicable(\Pi, C, asyn) = \left\{r_1, r_1^2, r_1^3, r_2, r_2^2, r_1r_2, r_1^2r_2\right\}$. The maximally parallel set of multisets of rules is the following:

$$Applicable(\Pi, C, max) = \left\{r_1^3, r_2^2, r_1^2r_2\right\}.$$

The application of the multiset of rules $r_1^2r_2$ on C yields $(1, a^4b^4c^3)$.

Example 3.3. Consider the system $\Pi = (O, w_1, w_2, R)$, with $O = \{a, b, c\}$ and $R = \{r_1 : (1, a)(2, b) \to (1, b)(2, a); r_2 : (1, b) \to (2, b)\}$. Consider the configuration $C = (1, a^4)(2, b)$. It is easy to observe that at each step only a single rule is applicable (r_1 or r_2 alternatively). The only possible sequence of rule applications is $(r_1r_2)^4$ after which no rule is applicable anymore. This sequence moves all symbols a from cell 1 to cell 2. We remark that the above rules do not rewrite objects, but only move them in the structure.

3.2 Examples of P Systems

Here, we give the informal description of some main variants of P systems that were targeted for an implementation.

Transitional (cell-like) P systems [77]. This model considers that cells are organized in a tree structure and uses rules of following type: $(i, u) \to (i, u')(j, u'')(k_1, v_1) \dots (k_m, v_m)$, where j is the parent of i and i is the parent of $k_1, \dots, k_m$, $m \geq 0$. When $|u| > 1$ corresponding rules are called *cooperative*.

Symport/antiport P systems [4]. This variant considers that objects are not transformed but rather moved in a tree-like or graph-like structure. This corresponds to rules of form $(i, u)(j, v) \to (i, v)(j, u)$ or $(i, u) \to (j, u)$.

Populational Dynamics P (PDP) systems [13, 92]. The structure of this model corresponds to several trees that have their roots linked together. There are two types of rules: (1) working inside some tree: $(i, u)(j, v) \to (i, u')(j, v')$, where i is parent of j and (2) communication between tree roots: $(r_i, x) \to (r_1, x_1) \dots (r_k, x_k)$, where $r_1, \dots, r_k$ are the numbers of the corresponding tree roots. Moreover, each rule has a probability associated to it, so the derivation mode is maximally parallel, followed by a probabilistic choice between rules having the same left-hand side.

Spiking Neural P systems [24, 49]. This model has a graph structure and restricts the alphabet of the system to be a single letter, however, permitting and forbidding conditions are replaced by a regular expression check. The rules are of form $(i, a^k) \to (k_1, a^{x_1}) \dots (k_m, a^{x_m}); (i, E)$, where E is the regular expression checking for the contents of cell i, a is the single symbol used in the alphabet and $k_1, \dots, k_m$ are the cells linked to cell i. The derivation mode is sequential at the level of each cell (only one rule per cell is applied) and maximally parallel at the level of all cells.

(Enzymatic) Numerical P Systems [78, 86]. A multiset M can also be seen as a function $M : V \to \mathbb{N}$. The model of numerical P systems extends the notion of multiset to $M : V \to \mathbb{R}^n$. Hence, the

configuration is a vector of real-valued variables and rules are equations describing how to update each variable during each step.

P Systems with Active Membranes [36]. This model corresponds to transitional P systems with an additional rule allowing to create new child cells.

4 REDUCTION OF THE COMPUTATION STEP TO MULTI-CRITERIA OPTIMIZATION

The computation in P systems corresponds to a finite sequence of transitions between configurations. Each transition $C \Rightarrow C'$ can be seen as a sequence of four steps: (1) computing the set of multisets of applicable rules ($Applicable(\Pi, C, asyn)$), (2) restricting it according to the derivation mode δ ($Applicable(\Pi, C, \delta)$), (3) choosing one element from it R, and (4) applying R to the configuration C yielding $C' = Apply(\Pi, C, R)$. The first step is the most time-consuming, as the corresponding problem is generally NP-hard. As we show below, in some cases it is possible to reduce the first three steps (the computation of an element from $Applicable(\Pi, C, \delta)$) to a multi-criteria optimization problem. This will allow us to express different algorithms used to compute this step by different authors in a common language for a better comparison. Another theoretical advantage of such reduction is an extensive standard vocabulary and a plethora of solving methods existing in the optimization area, which could be possibly applied for large-scale P systems. We would like to remark one more time that the presented reductions are not possible in the general case, however, they are possible for many concrete cases.

4.1 Preliminaries

A *multi-criteria* optimization problem (MCOP) is an optimization problem that involves multiple objective functions. In mathematical terms it can be stated as

$$\begin{aligned} &\max(f_1(x), f_2(x), \ldots, f_m(x)) \\ &\text{subject to } x \in X, \end{aligned}$$

where $m \geq 2$ and X is the set of *feasible* vectors (or solutions). This set is usually defined by some constraint functions. We can also consider the objective function as a vector: $f : X \to \mathbb{R}^m$, $f = (f_1, \ldots, f_m)$. For a feasible solution x, the vector $z = f(x)$ is called an objective vector or an outcome [50, 53].

When corresponding functions as well as $f_i, 1 \leq i \leq m$ are linear, we speak about a multi-criteria *linear* optimization problem. We also remark that as for classical optimization problems the objective functions are minimized; the other cases such as maximization or hybrid min/max can be easily reduced to the minimization one. When further $X \subseteq \mathbb{N}^k$, $k > 0$, and $f : X \to \mathbb{N}^m$, we speak about an *integer* multi-criteria linear optimization problem (IMCLOP).

In multi-criteria optimization, typically there is no feasible solution that minimizes all objective functions simultaneously. Hence, the main attention is focused on solutions that cannot be improved in any of the objectives without degrading some other objective(s). Such solutions are called *Pareto-optimal*. Formally, they are defined as preimages of maximal elements of the outcomes, which are also called *Pareto front*.

Definition 4.1. A vector $x \in X$ is Pareto-optimal for a MCOP (defined as above) iff there is no other vector $y \in X$ for which $f(x) < f(y)$, where $u < v$ iff $u_i \leq v_i,\ 1 \leq i \leq k$ and $\exists j, 1 \leq j \leq m$ such that $u_j < v_j$.

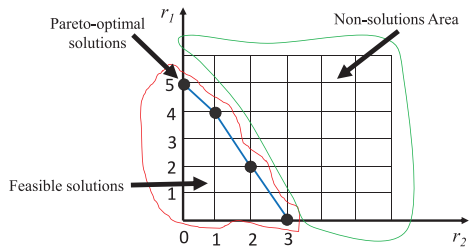


Fig. 1. Feasible and Pareto-optimal solutions from Example 4.2.

Example 4.2. Consider the following problem:

$$\begin{aligned} &\max(r_1, r_2) \text{ subject to} \\ &r_1 \leq 5 \\ &r_1 + 2r_2 \leq 6 \\ &r_2 \leq 3 \\ &r_1 \in \mathbb{N}, \quad r_2 \in \mathbb{N}. \end{aligned}$$

The corresponding feasible solutions and Pareto-optimal solutions are shown in Figure 1. We note that Pareto-optimal solutions are (5,0), (4,1), (2,2), and (0,3).

4.2 Rule Choice as IMCLOP

It is not difficult to see that the problem of the computation of elements from $Applicable(\Pi, C, max)$ can be reduced to IMCLOP. For the first time it was noticed in Reference [5], but without any further development. For simplicity, we consider that Π is a transitional P system and has only one membrane (and no environment). If this is not the case, we can apply the flattening procedure reducing it to one membrane [33, 35]. So $\Pi = (O, w_1, R)$.

Let $R = \{r_1, \ldots, r_m\}$ and $O = \{a_1, \ldots, a_n\}$. Consider that $r_i : (1, u_i) \to (1, v_i), 1 \leq i \leq m$. Let C be the current configuration and let $C_a = |C|_a, a \in O$.

We will consider a set of variables x_i, $1 \leq i \leq m$, which indicate the cardinality of corresponding rules in some parallel multiset $\mathcal{M} \in Applicable(\Pi, C, asyn)$, $|\mathcal{M}|_{r_i} = x_i$, $1 \leq i \leq m$. Then the feasible set X of (asynchronous) solutions is defined by the following inequalities:

$$\sum_{i=1}^{m} |u_i|_a x_i \leq C_a, \quad \forall a \in O, \tag{1a}$$

$$x_i \in \mathbb{N}, \quad 1 \leq i \leq m. \tag{1b}$$

Inequalities (1a) state that the sum of all consumed objects is included in C. Technically, for each object $a \in O$ it is verified that the weighted sum (by the number of symbols a in the left-hand side) of rule cardinalities is smaller or equal than the number of objects a in C. We remark that system (1) can also be seen as a system of Diophantine equations and corresponding solutions are exactly describing the set X of feasible solutions.

The IMCLOP corresponding to the computation of $Applicable(\Pi, C, max)$ is defined by:

$$\begin{aligned} &\max(x_1, \ldots, x_m) \\ &\text{subject to } (x_1, \ldots, x_m) \in X. \end{aligned} \tag{2}$$

It should be clear that Pareto-optimal solutions represent exactly the multiplicities of rules for some maximally parallel solution.

Example 4.3. Consider the system $\Pi = (O, w_1, R)$, with $O = \{a, b, c\}$ and $R = \{r_1 : (1, ab) \to (1, abc); r_2 : (1, bbc) \to (1, abb)\}$. Consider the configuration $C = (1, a^5b^6c^3)$. Then, the constructed IMCLOP corresponds to the one given in Example 4.2

So this system has four Pareto-optimal solutions: $(5, 0)$, $(4, 1)$, $(3, 3)$, and $(0, 3)$, corresponding to multisets of rules r_1^5, $r_1^4 r_2$, $r_1^2 r_2^2$, and r_2^3, which are exactly the maximal multisets of rules applicable to C.

4.3 Tentative Solutions

One of the traditional approaches to solve multi-criteria optimization problems is called *scalarization*. It consists in reducing the corresponding MCOP to a single objective optimization problem by using a real-valued scalarizing function involving the objective functions and additional scalar or vector parameters and variables. This can also imply additional restrictions to the feasible set based on the newly introduced variables.

One of the "simplest" methods to solve multi-criteria problems is the weighted sum method, where we solve the following single objective optimization problem:

$$\max_{x \in X} \sum_{k=1}^{m} \lambda_k f_k(x). \tag{3}$$

The weighted sum problem (3) is constructed using the scalar vector product of the vector of objective functions f and the vector of non-negative weights $\lambda \in \mathbb{R}^m$ as a parameter. It is known that it allows to compute all Pareto-optimal solutions for convex problems by varying λ [27].

In the literature on P systems some simple variants of the weighted sum method can be found. In References [3, 89] the vector $\lambda = (1, \ldots, 1)$ is considered (so the objective function is the sum of all variables). However, in this case only maximally parallel rulesets having a maximal number of rules are obtained. In terms of the formal framework [35], this corresponds to $max_{rules}max$ derivation mode.

Another attempt was done in References [20, 21] where the parameters λ_k correspond to the size of the left-hand side of rules ($\lambda_k = |u_k|$, $r_k : u_k \to v_k$). Such optimization problem finds only maximally parallel solutions involving the maximal number of objects, corresponding to $max_{objects}max$ mode in terms of the formal framework.

In Reference [71] the set of maximally parallel multisets of rules is expressed as solutions of a system of Diophantine equations (roughly, Equations (1a)) with an additional constraint to be satisfied on a solution, expressed as another system of Diophantine equations.

Finally, in Reference [9] the set of maximally parallel multisets of rules can be expressed as solutions to a system of equations defining some Diophantine sets. While the construction is similar to the one we give below, it is not trivial to manipulate Diophantine sets and it is not clear how to express the constraints as a single system of equations. Below, we show how to handle this problem by using same construction as for handling multiple "either-or" constraints in integer linear programming.

As above, we consider that Π has only one membrane (and no environment). So, $\Pi = (O, w_1, R)$.

Let $R = \{r_1, \ldots, r_m\}$ and $O = \{a_1, \ldots, a_n\}$. Consider that $r_i : (1, u_i) \to (1, v_i), 1 \leq i \leq m$. Let C be the current configuration and let $C_a = |C|_a, a \in O$.

In addition to the λ vector above, we introduce an integer parameter $M \in \mathbb{N}$ having a value that is sufficiently big (in fact, it should be greater than the maximal possible value of any variable x_i, $1 \leq i \leq m$ of any feasible solution). We remark that in general case of maximal parallelism it is impossible to limit the maximal values of variables, however, for practical applications, it is often possible to find such a bound.

We will modify the system (1) to construct a system of inequalities whose integer solutions will be maximally parallel multisets of rules for the configuration C.

$$\sum_{i=1}^{m} |u_i|_a x_i \leq C_a, \quad \forall a \in O, \tag{4a}$$

$$\sum_{i=1}^{m} |u_i|_a x_i + |u_k|_a + M z_a^k \geq C_a + 1, \quad 1 \leq k \leq m, a \in O, |u_k|_a > 0, \tag{4b}$$

$$\sum_{a \in O} z_a^i = N_i - 1, \quad 1 \leq i \leq m, \; N_i = \sum_{a \in O} \operatorname{sgn}(z_a^i) \tag{4c}$$

$$x_i \in \mathbb{N}, \quad 1 \leq i \leq m, \tag{4d}$$

$$z_a^i \in \{0, 1\}, \quad 1 \leq i \leq m, a \in O. \tag{4e}$$

Inequalities (4a) are the same as (1a) and they state that the sum of all consumed objects is included in C.

Inequalities (4b) state the maximality property of the rule set defined by $x_1, \ldots, x_m$. It verifies that for each rule there exists at least one object whose remaining quantity is not sufficient to apply this rule. They are based on multiple "either-or" constraints representation in ILP. The big value of M and the inequalities (4c) ensure that only one constraint from (4b) will be considered (the other ones will be satisfied because of the big value of M).

Hence, any solution $x_1, \ldots, x_m$ satisfying the system of inequalities (4) corresponds to a Pareto-optimal solution of (3), hence, to a maximally parallel rule set $\mathcal{M} = r_1^{x_1} \ldots r_m^{x_m}$ applicable to configuration C. We also remark that from the construction given above, it immediately follows that system (4) is Diophantine.

Example 4.4. Consider the system $\Pi = (O, w_1, R)$, with $O = \{a, b, c\}$ and $R = \{r_1 : (1, abc) \to (1, ab); r_2 : (1, a) \to (1, bb); r_3 : (1, b) \to (1, cb)\}$. Consider the configuration $C = (1, a^2b^3c^2)$. Then, we construct an ILP according to the rules above (we recall that M is a big integer number):

Derived from (4a):

$$\begin{aligned} x_1 + x_2 &\leq 2 \\ x_1 + x_3 &\leq 3 \\ x_1 &\leq 2. \end{aligned}$$

Derived from (4b) for r_1:

$$\begin{aligned} x_1 + x_2 + 1 + M z_a^1 &\geq 3 \\ x_1 + x_3 + 1 + M z_b^1 &\geq 4 \\ x_1 + 1 + M z_c^1 &\geq 3. \end{aligned}$$

Derived from (4c) for r_1:

$$z_a^1 + z_b^1 + z_c^1 = 2.$$

Derived from (4b) for r_3:

$$x_1 + x_2 + 1 \geq 3.$$

Derived from (4b) for r_3:

$$x_1 + x_3 + 1 \geq 4.$$

It is not difficult to see that this system has three solutions: $(2, 0, 1)$, $(1, 1, 2)$, and $(0, 2, 3)$, corresponding to multisets of rules $r_1^2 r_3$, $r_1 r_2 r_3^2$, and $r_2^2 r_3^3$, which are exactly the maximal multisets of rules applicable to C.

We remark that all solutions of (4), (3) might be tedious to obtain. For the simulation purposes, only one such solution is necessary.

As previously mentioned, using the weighted sum scalarization technique it is theoretically possible to reach any single point from the Pareto front by choosing appropriate values of the parameter vector λ. However, by using Equations (4b) it becomes much easier to choose corresponding parameters, as the feasible set is restricted only to Pareto-optimal values. For example, using $\max x_1$ as objective function with the constraints from Example 4.4 would yield the solution $(2, 0, 1)$. By using $\min(x_1 + x_2 + x_3 - 5)$ as the objective function, the solution $(0, 2, 3)$ is obtained (the above constraint allows to consider only multisets of rules of size 5).

In the literature on P systems, there are other examples of the construction of a single Pareto-optimal solution having certain properties. One of them is the DND algorithm introduced in Reference [71] for the FPGA simulations that we discuss below.

In what follows, we explain briefly the functioning of the DND algorithm in the view of Equations (1). First a random rule permutation is computed (corresponding to a random permutation of variable indices). Next, during the forward step a random value (bounded by the number of possible applications) for the number of each rule applications is taken. This corresponds to finding the values of x_i, satisfying the constraints (1a). Finally, during the backward step, the frequency of each rule is increased until it cannot be applied anymore. This step corresponds to the elimination of the dominated, i.e., smaller, solutions for the system (1) yielding only Pareto-optimal ones.

The described procedure can also be seen as a combination of scalarizing and of the lexicographic method, which is a method from the family of *a priori* methods for multi-criteria optimization. More precisely, it corresponds to a solution of a series of optimization problems, each of them bounded by the parameter corresponding to the choice of the maximal rule multiplicity for the forward step. The backward step also corresponds to a series of optimization problems that just reach the maximum for each component (like in the lexicographical method).

A simpler variant of the DND algorithm that does not perform the initial rule permutation can be found in References [60, 98, 100]. Other variations of DND algorithm were used for CUDA-based simulations and are discussed later. A different approach is used in Reference [106]. It supposes that for a P system Π working in the derivation mode δ there exists an easily computable function $NBVariants(\Pi, C, \delta)$ that for any configuration C gives the number of solutions of Equation (4). Next, it also supposes that there exists an easily computable function $Variant(\Pi, C, \delta, n)$ that for each integer n (up to the corresponding value) yields the corresponding solution (the used method is similar to the decoding of a number in the combinatorial number system). It is clear that such functions do not exist for any variant of P system. References [88, 106] give some sufficient relations between rules that allow to construct the above functions for a P system working in set-maximal (*smax*) derivation mode.

5 IMPLEMENTATION OF P SYSTEMS USING HARDWARE

The discussion about hardware implementation of membrane computing concerns mainly the simulation of concrete variants (sometimes even examples) of P systems using a dedicated hardware (for ASIC and FPGA) or firmware (for CUDA). In most of the cases a (single) particular class of

P systems is simulated. The simulator is composed from two parts: (1) the hardware simulator for a concrete system and (2) the HDL code generator of hardware simulators, which based on input parameters (rules, membranes, initial configuration, etc.) generates the code for the dedicated hardware simulator. In this section, we will mainly discuss the structure of the corresponding hardware simulators, as the generator part is more or less following standard compiler construction techniques. We will concentrate on three points, which are the most important for a hardware implementation: (a) the representation of the configuration (multisets of objects and the membranes), (b) the representation of rules and their parallel application, and (c) handling of the non-determinism.

5.1 Data Organization

Membranes. There are no compartments in silicon circuits, hence, the notion of membranes is relatively difficult to represent directly. Nevertheless, according to Reference [85], a membrane is just an idealized concept without internal structures. The main functionality of membranes is to perform a topological division of the space allowing P systems to compute in a distributed manner (based on a correspondence between the membrane and its contents). Hence, the spatial placement and size of membrane are not important, only the inter-relationship among them matters. Moreover, it is known that any membrane structure can be reduced (flattened) to just a single membrane (see References [33, 35] for more details). The existing hardware simulators adapt in most cases this last point of view, where the membrane structure is not physically implemented on the device. As exceptions from this rule, we cite References [73, 82] (region-based), which implicitly implement the membrane topology by using dedicated buses and message passing in the corresponding circuits.

Configuration. The representation of the configuration in all cases is done as a vector of non-negative integers (stored in the memory/registers of the device). We remark that this vector corresponds to a flattened system, so it is relatively big and sparse, as it encodes each object/membrane pair. For performance reasons, it is physically split in several places to be closer to the processing units (although routing is relatively complicated). In the case of References [73, 82] it can be argued that corresponding parts are internalized into the corresponding region circuit, as the access to corresponding values is not direct and it is done by message passing. In CUDA, only the portions of the array representing the configuration to be processed are loaded by the cores.

Evolution Rules. As it can be seen from Definition 3.1 in the simple case (without permitting and forbidding) rules can be defined by two integer vectors indicating the multiplicities of corresponding objects in the left-hand side and right-hand side of each rule. This gives a natural rule representation as two vectors stored in the memory/registers. Then a specific circuit/module verifies the applicability of rules and performs their application. Most variants of hardware implementations use this idea, however, in References [70, 72, 82] each rule is encoded in hardware as a specific circuit that verifies the needed resources and performs the rule application. In the case of P systems with active membranes, there might be several cells with the same label, so the corresponding CUDA simulator uses a split representation: two integer vectors, as above, for all rules, and an index array with one position per rule giving for each cell the rules it contains.

5.2 Object Distribution Problem and Non-determinism

Hardware implementation of P systems faces the problem of computing the applicable rule set according to some derivation mode (usually maximally parallel). More precisely, an efficient way to compute and represent an element from $Applicable(\Pi, C, \delta)$ is required [88, 106]. The difficulty of this problem is that rules can compete for the same objects, so increasing the number of

occurrences for one rule may decrease the application possibilities for another one. Another important problem is to ensure that a non-deterministic choice among all possibilities is performed. In References [41, 44, 57, 60], hardware architectures aiming at parallel processing and communication, and the application of rules are developed. In Reference [11], a formal exposition of non-deterministic evolution in transition P systems was suggested.

We call the first problem as *object distribution problem* (ODP). It consists in the computation of the set $Applicable(\Pi, C, \delta)$ (or of an element from this set). As discussed in Section 4 in terms of multi-criteria optimization, this corresponds to the computation of the corresponding Pareto front (or an element of it).

In Reference [71] different algorithms solving ODP are classified in *direct* and *indirect* ones. In the direct approach, the corresponding multiset is directly constructed by the algorithm. In terms of MCOP this corresponds to a particular fixed scalarization. The indirect approaches are based on the observation that the solution number is finite, because the solution space is bounded by the size of the configuration. Hence, a heuristic or brute-force approach can be used to explore this bounded space. However, since it is an overestimation, there might be visited elements that are not valid solutions. Hence, the algorithms are iterative and explore the whole space until a valid solution is encountered. In terms of MCOP this corresponds to different searches through the space limited only by the maximal values for each axis.

Sometimes it is not easy to classify an algorithm in one of these categories. We will classify an algorithm as a direct approach if its main goal is to construct a valid multiset of rules. Otherwise, if an algorithm is exploring different solutions until it reaches a valid one, it will be classified as indirect. We will use this classification to overview different strategies for ODP solution known in the literature.

Indirect approaches. Generally, the enumeration of all possible solutions and their verification one-by-one until a correct solution is obtained is the simplest method for the indirect approach [31]. Before the first correct solution is obtained, some invalid solutions should be rejected. This approach is called *indirect straightforward approach* [71]. Taking into account that it is not viable to enumerate all possible solutions for many problems, the feasibility of the approach is low. However, the performance of the algorithm suggests its use as to compute the floor values for the object distribution problem. Another indirect approach discussed in References [71, 73] called *indirect incremental approach* investigates a strategy generating possible solutions in rounds. Other attempts based on a similar idea but with different rule elimination strategies were done in References [30, 39, 46, 48, 98, 99].

Direct approaches. In contrast to indirect approaches, the direct approach fabricates a solution straightforwardly rather than identifying a number of possible solutions before a solution is confirmed.

The simplest approach is the *direct straightforward approach*. As defined in Reference [71], in this approach "all the solutions to the object distribution problem are given as input, and one of these solutions is simply selected at random." While in the same paper it is argued that such approach is infeasible for an arbitrary configuration and rule types, it can still be applied in a large number of cases. As shown in References [88, 106], if at each step the number of solutions can be expressed as the number of words of some length in a regular language, then it becomes possible to compute the solution only based on its number. In Reference [106] it is shown that the corresponding class of P systems is quite large and also that this method is particularly interesting for bounded derivation modes like the set-maximal derivation mode (called also flat mode) where the rules are chosen in a set-maximal way (instead of the multiset maximal way).

Another variant of the direct approach is the *Direct Non-deterministic Distribution algorithm* (DND) proposed in Reference [71]. A similar algorithm can also be found in References [38, 40]. This algorithm works in two phases. At the first phase all rules (initially randomly shuffled) except one are selected to be applied a random number of times below its maximal applicability value. In the second phase, all the rules are taken in the converse order and their applicability is increased up to the maximal still possible value, except that the last rule in the first phase keeps its original applicability value. A variant of DND, named DND-P, became popular in the simulation of Population Dynamics P (PDP) systems [65]. Together with another algorithm, Direct distribution based on Consistent Blocks Algorithm (DCBA) [63], it was employed for the engine of the PDP system simulator on CUDA [62].

Non-determinism. One of the difficulties of the above approaches is the handling of non-determinism. From the formal point of view, the non-determinism corresponds to a random equiprobable choice of an element from the set of all applicable multisets of rules ($Applicable(\Pi, C, \delta)$). In the case of indirect approaches, due to the iterative nature of the algorithms, it is not easy to argue that each possibility has the same probability to occur. We would state that solutions containing a smaller number of different rules have a higher chance to be selected. In the case of DND algorithm and related variants, it looks like that the obtained solution tends to be an equiprobable choice. However, the corresponding articles do not give such a proof and there are some unclear points, which do not allow us to affirm this fact. Up to now, the only algorithm that is performing a truly non-deterministic choice is the one described in References [88, 106]. However, the corresponding implementation is limited to some particular derivation modes and particular classes of P systems.

6 AN OVERVIEW OF EXISTING FPGA SIMULATIONS

With the advent of reconfigurable hardware that realizes the idea of modifying the hardware circuits by programming, conceiving a novel circuit simulating an innovative processing paradigm is no longer an exceedingly hard task. The first attempt to use FPGA reconfigurable hardware to simulate P systems dates back to 2003 [82]. Since then, two simulation approaches emerged, considering regions or rules as basic processing units.

6.1 Region-based Simulations

In the region-based simulation approach rules and objects from different membranes are physically located in different places of the circuit, while those from the same membrane are physically close and well connected. The biggest problem is to ensure the correct communication of objects between membranes, as this requires a global-level synchronization. As advantage, the obtained system is highly scalable and robust. Below, we give two examples of region-based simulations.

In contrast, the rule-based implementation approach discussed later explicitly represents the evolution rules as processing units and multisets of objects as register arrays, while membranes and regions are represented implicitly as logical constructions existing between those processing units and data structures.

6.1.1 Petreska and Teuscher Simulation. Petreska and Teuscher designed the first FPGA circuit simulating membrane computing (more precisely, transitional P systems) [82]. They could realize several interesting features such as communication to inner membranes, priorities between rules, and proposed ideas how to simulate membrane creation and dissolving mechanism in integrated circuits. This work inspired the successors to engage in this challenging and breathtaking field to advance the development of hardware simulation of P systems. In their simulation they made two strong assumptions: the application of evolution rules in each membrane is not done in a maximally

parallel but in a sequential manner (but still keeping a parallelism at the system level); the non-deterministic evolution of configuration is substituted by a deterministic transition following a predetermined order. Such computational step corresponds to an ILP with the subject function as a weighted sum of variables with predefined fixed weights.

In theory, membranes are borders without internal structures and material consistence. In this implementation, membrane structures represent regions on the circuit containing the enclosed substances, i.e., the multisets of objects, evolution rules, and children membrane architectures. The objects exchange among membranes is a kind of bi-directional traversing behavior. In case of the possible objects exchange between regions, the communications are made by data buses connecting to different parts of hardware representing inner membranes. To avoid the multiple buses used to connect the parent membrane to its plural children membranes, a single bus links all the children membranes before it connects to the parent membrane. Hence, the communication is limited to parent-child membranes and there is no object exchange among children membranes or non-immediate contained membranes.

The representation of the multisets of objects is implemented by using registers. Different registers just preserve different multiplicities of objects. A register does not store the objects but only a number indicating the multiplicity of each object. The order of these registers is in accordance with the lexicographic order of the alphabet of objects. The recognition of an object is indirectly realized by examining the position of the register storing the multiplicity of this object. An evolution rule defined here is in the form of $u \rightarrow v(v_1, in_i)(v_2, out)$, where v_1 is the string to be sent into lower-immediate membrane labeled i, v_2 will be sent to upper-immediate membrane. The treatment employed to deal with the formulation of evolution rules is storing the rule's left-hand side and right-hand side into different registers separately. A particular module is designed to determine whether a rule is applicable. This module compares the left-hand side of a rule u with the multiset of objects w present in the current membrane. If and only if $u \leq w$, this rule is applicable and this module will generate a signal $Applicable = 1$. Input all the *Applicable* signals to an OR gate, the result of this logical gate can used as a monitor to identify whether the evolution reaches halt configuration.

The transition of configurations of P system is realized deterministically and sequentially, which is different from the general model. The consecutive transformation of configurations is regarded as the evolution process. This evolution process is decomposed into micro-steps and macro-steps. The application of rules enclosed by membranes is performed in terms of a predefined sequential order. This deterministic execution of rules is conducted in micro-steps sequentially. If a selected rule is applicable, the left-hand side of the rule u will be removed. Then the right-hand sides v, v_1 and v_2 are stored in corresponding registers. Objects from the upper immediate membrane will be preserved in another register. Although the micro-steps are carried out deterministically, they are performed simultaneously in all membranes until there are no applicable rules. The micro-steps terminate when there are no applicable rules, i.e., the halt condition is reached. All the registers are updated in line with associated rules in macro-steps.

This implementation considered and respected the priorities of applicable rules at the beginning of each micro-step. By labeling the applicable rules with higher priorities and storing the corresponding labels, applicable rules are executed in accordance with their respective priorities. Besides, two additional features of P system, the dissolution and creation of membranes, are simulated. When a rule with membrane dissolving function is applied, its contents are owned by its upper immediate membrane, setting the membrane *Enable* signal of the relevant membrane to "0." However, the connections and registers defining the dissolved membrane still exist. This scheme gives rise to a disadvantage that the hardware resources cannot be released. The creation of new membranes is executed in the initialization process of the P system, since all the information about

new membrane is known from the specification of the system. The created membranes are inactive until membrane creating rules invoke them.

6.1.2 Nguyen Simulation. In this implementation, a parallel computing platform simulating membrane computing based on FPGA named Reconfig-P is developed [69, 74]. Reconfig-P is fabricated on the basis of the region-oriented idea that regions work as the computational entities communicating objects through message passing. The functionality of these regions is extended by the included set of evolution rules. P Builder, the software component of Reconfig-P, specifies the P system concerned in software, converts the specification of P system written in Java to Handel-C (a hardware description language) source code. Software simulation of the circuits to be constructed is supported by P Builder to test the functionality of circuits before mapping the code to hardware circuits.

The execution of a evolution step is divided into two phases: *object assignment phase* and *object production phase* [69]. The maximal instance of each rule in a region is determined in the object assignment phase. The update of multiplicity of objects is accomplished in the object production phase. The maximal instance of the rules with higher priorities is computed before the rules with lower priorities. Note that the consumption of objects for rules with higher priorities performed during the object assignment phase to save clock cycles. It is assumed that all rules are assigned relative priorities. The priority between rules is implemented as the temporal order, which should be respected by region processing units in the assignment phase. Rules with same priority are executed concurrently. The temporal order is determined at compile-time. Rules are applied according to their priorities in rounds until no rules are applicable using the indirect iterative approach. Under this circumstance, the applicability of each rule is non-stationary because of the existence of priorities. To avoid processing inapplicable rules, the applicability status of each rule is checked at the outset of the assignment phase and immediately after an applicable is applied to consume some objects.

Objects traversing behavior is the origin of communication between regions. The update of multiplicity of objects caused by rules with and without traversing behavior is completed in the object production phase. When different region processing units update the multiplicity value of the same object at the same time, a conflict occurs. To handle this conflict, in Reference [69] two solution strategies, the *space-oriented strategy* and the *time-oriented strategy,* are proposed. Tables 1 and 2 summarize the different strategies of two resource conflict resolutions and their modifications in the rule-based and region-based design [70, 72]. To simplify the exposition of processes of rule-based and region-based implementations of P systems, tables are designed to delineate the relevant details, which will be given below.

The extensibility of the region-based design is the consequence of the representation of membranes as processing units interacting with two region processing units corresponding to inner and outer regions. This allows to achieve a strong separation of the processing logic inside different membranes and the independence of the communication. Thus, adding additional elements to the system does not lead to the redesign of the remaining part of the system.

6.2 Rule-based Simulations

Rule-based approaches consider evolution rules as processing units performing the update of multiplicities and of membrane structures.

6.2.1 Nguyen Simulation. Every rule in all regions of the P system is represented as a processing unit synchronized by a global clock that implements the parallel processing. In the design phase, a processing unit corresponds to a potential infinite *while loop* that contains the Handel C codes related to the rule application. The information associated to execution and

Table 1. The Comparison of the Time-oriented and Space-oriented Conflict Resolution

Strategy	Resource conflict resolution
Time-oriented strategy	(a) Construct a conflict matrix in which each row is a quadruple (p, q, r, s). p is the object competed by multiple rules. q is the region where p is produced or consumed. r is the set of the conflicting rules, s is the size of set r. (b) Insert delay statement among conflict rules such that the updating operations of multiplicity of p can be executed in distinct clock cycles. The number of delays is equal to s.
Space-oriented strategy	(a) Construct the conflict matrix as in the Time-oriented strategy. (b) The register storing the object accessed by multiple rules concurrently is replicated s times. These copy registers are assigned to each conflicting rules to write. After the updating process for all the copy registers, the corresponding values are joined to the original register.

Table 2. The Differences of the Conflict Resolutions Adopted in Two Design Modes

Item	Time-oriented strategy	Space-oriented strategy
Rule-oriented design	The interleaving operation can be determined at compile-time and it can be hard-coded into the HDL source.	Need a multiset replication coordinator to coordinate the multiplicities stored in the copy registers.
Region-oriented design	The objects received from other regions are regarded as *external objects*, otherwise the objects are *internal objects*. The interleaving merely caused by the production of internal objects can be identified at compile-time. To retain the independence of region processing units, the interleaving induced completely or partially by the receipt of external objects can only be calculated at run-time.	The role of the multiset replication coordinator in rule-oriented design is played by the region processing units. The existing register storing the multiplicity received from the associated communication channels in the considered region can be assigned to those processing units that send objects to the considered region to write the new values of the objects competed by multiple rules.

synchronization is contained in processing units as well. Each rule processing unit in a region is linked to the array of registers containing multisets of objects. The membrane inclusion relationships can be described with the connections between processing units and arrays. Generally speaking, a rule processing unit in a region is linked to the objects array located in the same region. If there are objects traversing regions that imply the containment, connecting the rule processing unit to the object array to which the rule will send objects contained in different region. This procedure permits to represent the membrane inclusion. The rule execution is split into *preparation phase* and *updating phase*. We compare the two designs (the rule-based and the region-based) of Nguyen's implementation in Table 3.

6.2.2 Verlan and Quiros Simulation. The target model is a static P system. The system is considered flattened, so only one skin membrane is present. A special strategy was elaborated to not compute the complete solution, i.e., $Applicable(\Pi, C, \delta)$ but the cardinality of its elements. Then a random value between 1 and this cardinality is taken. Finally, this number is decoded to the corresponding solution [88, 106].

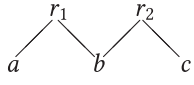
Devising an algorithm that carries out the computation of the cardinality and of all elements of the solution set in constant time on FPGA is the key issue of the approach. A remarkable

Table 3. The Comparison of the Rule-based and Region-based Design of Nguyen's Implementation

Object	Rule-based Design	Region-based Design
Region and their containment relationships	Regions are realized in hardware implicitly by the content they included. For the containment relationships including the traversing of objects between regions, they are implemented by imparting the corresponding rules with "in" or "out" target directives the abilities to access the multiset of objects in the destination region of the objects traversed.	Regions are represented as parallel processing units. The traversing of objects among regions is realized as message passing through channels connecting different region processing units.
Multiset of objects	An array of registers contains each type of object in the alphabet for a region	The same strategy adopted as the rule-oriented design.
Evolution rule	Potentially infinite *while loop* in which contain procedures representing the operations of the relevant evolution rules in HDL language.	Implicitly expounded through integrating them into the region processing units.
Operation process	Preparation phase and updating phase.	Object assignment phase and production phase.
Synchronization	An array of registers composed of three 1-bit registers is associated to each rule processing unit. The values in the array indicate whether the rule processing unit has complete the preparation and updating phase, or is applicable. Compute the logic AND or OR values of all the values stored in the 1-bit register that indicates the same status of the processing unit and store the three results into three sentinel registers. The coordinating processing unit read the sentinel values to synchronize the whole procedures.	For the synchronization of object assignment phase, it is realized when all the region processing units communicate with each other on channels at the beginning of object production phase. For the object production phase, a region execution coordinator connecting to each region processing unit via dedicated channels is designed to perform the synchronization of operations. After the region execution coordinator received the signals denoting the completion of all the operations of every region processing unit, the current transition is done and the next one is carried out.

characteristic of FPGA is that the time consumed for executions of functions that do not exceed the cycle of the global clock is done in one cycle of FPGA, hence, in constant time. The computation of the cardinality and the decoding of solutions are accomplished by two functions hardwired into the circuit: $NBVariants(\Pi, C, \delta)$, which gives the cardinality of $Applicable(\Pi, C, \delta)$, and $Variant(n, \Pi, C, \delta)$, which returns the nth element of $Applicable(\Pi, C, \delta)$.

A concept named rules' dependency graph is introduced to compute the two functions above. It is a bipartite graph that contains as nodes rules and objects and there is an edge between two nodes if a rule contains the corresponding objects in its left-hand side. The picture below depicts the rules' dependency graph for rules $r_1 : ab \rightarrow u$ and $r_2 : bc \rightarrow v$.



Assume that the derivation mode is maximal parallelism (max). Suppose that N_a, N_b, and N_c represent the number of objects a, b, and c in C. Let $N_1 = min(N_a, N_b)$, $N_2 = min(N_b, N_c)$, $N = min(N_1, N_2)$, $k_i = N_i \ominus N, 1 \leq i \leq 2$, where $\ominus$ denotes the positive subtraction. Let also

$p, q = 0, 1, 2, \ldots, N$. From the dependency graph, we can deduce the following:

$$Applicable(\Pi, C, max) = \bigcup_{p+q=N} \left\{ r_1^{p+k_1} r_2^{q+k_2} \right\}$$

$$NBVariants(\Pi, C, max) = N + 1$$

$$Variant(n, \Pi, C, max) = r_1^{N-n+1+k_1} r_2^{n-1+k_2}.$$

Example 6.1. Consider a configuration where $N_a = 5$, $N_b = 5$, and $N_c = 3$. It can be easily verified that $N_1 = min(5, 5) = 5$, $N_2 = min(5, 3) = 3$, $N = min(5, 3) = 3$, $k_1 = N_1 \ominus N = 5 - 3 = 2$, $k_2 = N_2 \ominus N = 3 - 3 = 0$. Hence, we can enumerate the elements of $Applicable(\Pi, C, max)$ as below:

$$Applicable(\Pi, C, max)_1 = \{r_1^{3+2} r_2^{0+0}, r_1^{2+2} r_2^{1+0}, r_1^{1+2} r_2^{2+0}, r_1^{0+2} r_2^{3+0}\} = \{r_1^5, r_1^4 r_2, r_1^3 r_2^2, r_1^2 r_2^3\}.$$

The same result can be easily obtained by using formal power series associated to context-free languages. In this case, any maximal rule combination is a part of the language $L_N = \{r_1^p r_2^q \mid p + q = N\}$. It is quite easy to observe that the number of words of length N in L_N is exactly the same as the number of words of same length in the language $L = \{r_1^* r_2^*\}$. This last language is regular and its generating function is $q_0(x) = 1/(1 - x)^2$. The nth coefficient of the expansion of $q_0(x)$ is equal to $n + 1$ ($[x^n]q_0 = n + 1$), which immediately gives $NBVariants(\Pi, C, max) = N + 1$. The $Variant\ (n, \Pi, c, max)$ is computed using an algorithm that performs a weighted breadth-first search of the decomposition of n with respect to the number of variants found on each branch of the execution of automaton for L.

Such a process can be easily repeated for any regular language, yielding a constant time simulation of a computational step. The reason for such performance is that any generating function is equivalent to a recurrence relation and such relations can be computed in one synchronous time unit using asynchronous operations. The described algorithm functions for any P system where the rule choice can be expressed as words of certain length in a regular language. The corresponding class is quite large (containing even computationally complete models), thus allowing an extremely fast execution. Examples from Reference [106] exhibited a speedup of order 10^5.

Another important point is that this approach allows to handle the non-determinism in a natural way by performing a uniform random choice between all possible rule applications at each step.

Technically, the implementation represents only objects by registers and rules by layered logic. Each rule implementation is modularized and contains an own copy of processing instructions needed to compute the two above functions, based on asynchronous operations. Consequently, five clock cycles are required to compute the $NBVariants(\Pi, C, max)$, $Variant\ (n, \Pi, C, max)$ and to apply the corresponding rules. The entire process of the implementation is split into several consecutive stages, which take charge of different operations associated to phases of evolutions of configurations.

Persistence stage stores the states that the hardware system goes through. An independent stage computes the maximal instance of each rule by means of the dividing operation and MIN logic operation. *Assignment* stage is in charge of selecting a rule to be applied non-deterministically and determines its instances. *Updating* stage is responsible for updating the current configuration with the values from the previous stage. During *Halting* stage, the system inspects whether the halting condition is reached, and once reached, stops the system.

In order to more clearly show the FPGA implementation methods of P system proposed by the above three research groups, their methods are summarized and compared from the quantitative and qualitative perspectives in Table 4 and Table 5.

Table 4. The Quantitative Attributes of FPGA Implementation

Item	Biljana Petreska	Van Nguyen	Juan Quiros
Period	2003	2007–2010	2012–2015
Institute	Swiss Federal Institute of Technology Lausanne (EPFL)	University of South Australia	University of Seville
Target FPGA	Xilinx Virtex-II Pro 2VP50ff1517-7	Xilinx Virtex-II XC2V6000-ff1152-4 (rule-oriented) and Virtex-II RC2000 (region-oriented)	Xilinx Virtex-V XC5VFX70T and Virtex-VII XC7VX485T
Host processing platform	Not given.	1.73 GHz Intel Pentium M processor with 2 GB of memory	Intel Core i5—5220 at 3 GHz, with 8 GB of RAM.
HDL	VHDL	Handle C	VHDL
Experiment Subjects	Cell-like P systems with following characteristics: membrane dissolution and creation, objects exchange between upper- and lower-immediate membranes, cooperative P systems with priorities. The range (non-continuous) for object number is 6 to 12 and for membrane is 10 to 20.	For rule-oriented design, the subjects are cell-like P systems cascaded in vertical, horizontal, vertical, and horizontal structures. For region-oriented design, the objects are cell-like P systems containing hierarchical regions and tissue-like connected regions. The rule range is [10,50], [1,25] for regions, [3,200] for objects. The extent of the inter-region communication is [80,319].	The subject P system is simplified according to the multiset rewriting point of view, which it has a skin membrane, no inner regions. The four subjects differ in the rule dependencies that form chains: circular, 2-circular, linear, opposite. The object number range is [10,200].
Experiment results	The hardware consumption ranges from 4.2% to 33% (CLB). The extent of clock rate is 27 to 198 MHz.	For rule-oriented design, the number of rules applied per second ranges from 2.7×10^5 to 1000×10^5. The hardware consumption extent is 1.55% to 21.43% (LUTs). For region-oriented design, the hardware usage ranges 1.82% from 16.79%. The clock rate fluctuates from 52.63 MHz to 81.77 MHz. The biggest size that can be executed is a P system with 550 rules, 1,280 communication channels, 1,100 objects conflicts.	The hardware consumption ranges from nearly 1% to 48% (LUTs) or nearly 1.1% to 11% (slice). The period needed to perform a computation step fluctuates from 5.46 *ns* to 9.14 *ns*. The highest frequency exceeds 100 MHz, permitting 2×10^7 computational steps per second. The run-time for each experiment subject ranges from $3.017 \times 10^{-5} s$ to $4.174 \times 10^{-5} s$.
Parallelism	System-level parallelism	Region-level and system-level parallelism	System-level (there is only a skin membrane, so it is also region-level parallelism).
Non-determinism	No non-determinism	DND algorithm	True non-determinism.

7 AN OVERVIEW OF EXISTING CUDA SIMULATIONS

In Reference [61], it is concluded that the GPU is a suitable platform to accelerate the simulation of P systems because of the following features:

- *Good performance*: for example, the NVIDIA Tesla K40 delivers 1.43 TeraFLOPS double-precision peak floating point performance, 4.29 TeraFLOPS of single-precision, and 288 GBytes/s of global memory bandwidth;
- *An efficiently synchronized platform*: GPUs implement a shared memory system, avoiding communication overload;

Table 5. The Qualitative Attributes of FPGA Implementations

Item	Biljana Petreska	Van Nguyen	Juan Quiros
Membranes (regions) and their containment	Implicitly represented by the contents enclosed by the membranes. The objects exchange is interpreted as transferring objects with communication buses by which connect the origin region to the destination region.	For the rule-oriented design, the treatment is similar with Petreska's. For the region-oriented design, regions are represented as parallel processing units. The objects exchange among regions is realized as message passing through channels connecting different region processing units.	According to the multiset rewriting system framework, the topology structures of membranes are not important. What is concerned is the rule dependency graph.
Multiset of objects	Store the multiplicity value of each type of object in different registers whose positions indicate the type.	The same method.	The same method.
Evolution rule	Store the left-hand side and right-hand side of a rule in different registers.	In rule-oriented design, they are characterized as potentially infinite *while loop* in which contain procedures representing the operations of the relevant evolution rules in HDL. In region-oriented design, they are implicitly expounded through integrating them into the region processing units.	The logic of rules is distributed along the hardware components. There is no explicitly correspondence between rules and hardware components.
Operation Process	Micro-step: only one applicable rule is applied with respect to the instance number in a region. Macro-step: execute a micro-step concurrently in every region.	In rule-oriented design, perform preparation phase and updating phase. In region-oriented design, perform object assignment phase and production phase.	1. Persistence stage; 2. Independent stage; 3. Assignment stage; 4. Application stage; 5. Updating stage; 6. Halting stage.
Extensibility	Membrane-mediated features cannot be added in, since membranes are implicitly represented by their contents.	Because of the region-oriented design, Membrane-mediated features such as *symport* and *antiport* functions can be extended.	This implementation is designed only for P systems whose applicable multisets of rules can be represented as regular language. So its extensibility is limited.
Scalability	With the increase of the rules, the hardware usage rises approximately proportional. The clock rates decline a small amount. The membrane creation ability cripples clock rate significantly.	The hardware consumption scales linearly with respect to the size of the P system executed in both rule-oriented and region-oriented design. The performances grow linearly when the number of rules increase.	The hardware usage is scalable, but the rate of increase in the performance is not always linear for different systems with distinct structures.
Contributions	Membrane creation and dissolution functionality	Two kinds of methodology for P system characterization and resource conflict resolution, DND algorithm.	Put up with a new methodology with absolute equiprobability to implement nondeterminism.
Drawback	Partially parallelism, no nondeterminism	The equiprobability of DND algorithm has not been proven theoretically.	The types of P systems that can be implemented are confined to those whose rules can be represented as regular language.

- *A medium scalability degree*: The amount of resources depend on the GPU model, e.g., a K40 includes 2,880 cores and 12 GBytes of memory. If the resources of a GPU are not enough, there are more scalable solutions such as multi-GPU systems, but they then require communication among nodes;
- *Low-medium flexibility*: Although CUDA programming is based on C++, and hence programmers are free to use the same data structures than in CPU, both the algorithm and the data structure have to be adapted for best performance on GPUs.

Simulation algorithms implemented on CUDA have a common structure, in which for each simulated computation step, first the rules are selected (obtaining a multiset of rules, while consuming the left-hand sides), and secondly the rules are executed (based on the obtained multiset of rules, and generating the right-hand side. This strategy is necessary to synchronize which rules get executed in the corresponding computation step. For some models—for example, for those ad hoc simulators—the selection step is done in micro-stages.

In the following subsections, the existing CUDA simulations are summarized by organizing into P system models.

7.1 Cell-like P Systems

The first test of concept for simulating P systems on GPUs was applied to P systems with active membranes using CUDA [18]. This simulator performs only one computation out of the whole tree to avoid non-determinism by requiring the confluence property to the simulated P systems. Bearing this in mind, the "lowest-cost computation path" is selected: the one in which least membranes and communication are required. This is achieved by giving preferences to rules that lead to least membranes (e.g., dissolution over division rules). The simulation algorithm is composed of two main stages: selection and execution of rules. Selection is where the semantics of the model is actually simulated. Rules are chosen by following the defined constraints, altogether with a number of applications. The result of this stage is used for the next one, which is the execution of the rules; that is, updating the P system configuration. This two-staged strategy allows to synchronize the application of rules within and among membranes.

Both P systems and GPUs have a double-parallel nature [18], and this is harnessed for implementing a mapping: (elementary) membranes are assigned to thread blocks and a subset of rules to threads. Each thread is in charge of selecting rules for a portion of the defined objects in the alphabet. Note that this is enough given that in P systems with active membranes, rules have no cooperation. However, this mapping of parallelism is naive, since it assumes that all the objects in the alphabet can be present within each membrane. This requires allocating memory space and assigning resources (threads) to all of them. This, in fact, does not take place in the majority of P systems to be simulated, but turns out to be the smallest worst case to handle. Thus, the performance of the simulator completely depends on the P system being simulated and drops as long as the variety of different objects appearing in membranes decreases.

Non-determinism is handled by imposing the simulated P systems to be confluent; that is, all computations halt and they generate the same result [18]. This way, the simulator can choose any path in the computation tree, since the aim is to find a halting configuration. The GPU simulator takes advantage of this by preferring to select evolution rules rather than division or dissolution.

The performance of the simulator was analyzed on a GPU Tesla C1060 (240 cores, 4 GB memory) by using two benchmarks [61]: a simple test P system designed in a convenient manner (up to 7× of speedup), and a family of P systems designed to solve different instances of the SAT problem (1.67× of speedup).

Improvements to this design have followed [61], reporting up to 38× of acceleration when taking advantage of shared memory and data-transfer minimization. By constructing a dependency graph, the set of rules of the input model are arranged so that those having common objects in the left-hand side and in the right-hand side (respectively) are more likely to be in a node. This reduces communication given that thread blocks are assigned to nodes.

Another approach was to implement ad hoc simulators for a specific family of P system with active membranes allowing to solve SAT in linear time [19]. In this way, the worst case assumed before (all objects defined can appear in every membrane) is further reduced by analyzing the upper bound to the number of existing objects in membranes. In this way, the work done by threads is maximized, and the design remains similar: Each elementary membrane is implemented by a thread block, and each object of the input multiset is implemented by a thread. The experiments carried out on an NVIDIA Tesla C1060 GPU reported up to 63× of speedup. Further developments took place focusing on this family of simulators, with the aim at being better tailored to newer GPU architectures, and also to enable multi-GPU systems and supercomputers [19].

A related work was to explore which P system ingredients are better suited to be handled by GPUs. To this aim, another solution to SAT based on a family of tissue P systems having the operation of cell division was simulated [61]. The design is similar to the previous case: Each cell is simulated by a thread block, however, the constructed solution required a higher number of objects to be placed inside each cell. At the same time, this simulator does not need to store nor handle charges associated to membranes. Experiments on an NVIDIA Tesla C1060 GPU led to speedup by 10×. This showed that using charges associated to membranes helped to save instantiation of objects, and so, they entail a lightweight ingredient to be processed by threads.

7.2 Population Dynamics P Systems

Population Dynamics P (PDP) systems is a multi-environmental model successfully used for modeling real ecosystems. Thus, their efficient simulation was critical for experimental validation. Simulators for PDP systems typically run several simulations to extract statistical information from the models.

A CUDA simulator for PDP systems was presented in Reference [62]. The selected simulation algorithm was the DCBA [63], since it provides better accuracy in the simulation results. The selection of rules in DCBA consists of three phases: Phase 1 (distribution of objects), Phase 2 (maximality), and Phase 3 (probability). This algorithm uses a distribution table to distribute the objects in a proportional way between competing rules, i.e., with overlapping left-hand sides. Moreover, rules are grouped into *rule blocks* when having the same left-hand side.

It can be noticed that the approach of DCBA is to handle non-determinism by first using a uniform distribution of objects to competing rules. This requires an extra phase (2) to avoid rounding errors; in this way, a random order is employed to control the remaining rules and which one takes all objects. This phase is based on the procedure of DND.

The CUDA simulator for PDP systems [61, 62, 64] uses the following design: The contents of environments and rule simulations are fractioned and distributed all-over thread blocks, while individual rules are fractioned over threads. Phases 1, 3, and 4 are efficiently managed by the GPU, but Phase 2 can become a bottleneck. For Phase 3, a random binomial variate generation library was developed, so that binomial and multinominal distributions were supported for random number generation. In the benchmark using a set of randomly generated PDP systems (having no biological meaning), speedups of up to 7× were achieved on a Tesla C1060 with respect to a multicore version. The simulator was validated and tested by using a known ecosystem model of the Bearded Vulture in the Catalan Pyrenees, leading to speedups of up to 4.9× with a C1060 and 18.1× using a Tesla K40 GPU (2,880 cores). Finally, the simulator was extended in Reference [64]

in such a way that it introduces high-level information provided by the model designer into the code. This is accomplished through a new syntactical ingredient in P-Lingua 5 [80], called *feature*. This ingredient was employed to define the usual algorithmic scheme based on rule modules when designing biological models. This new version is called *adaptative simulator* and helped to obtain an extra speedup of 2.5× on a Tesla K40 and 1.8× on a Tesla P100 GPU (3,584 cores).

7.3 Spiking Neural P Systems

Parallel simulation of Spiking Neural P (SNP) systems has been based on a matrix representation so far [115]. The simulation algorithm uses the following vectors and matrices:

- *Spiking transition matrix*: stores information about rules and is employed for computing transitions. It assigns a row per rule and a column per neuron. The values coded correspond to the left-hand side of rules (as negative numbers) and to the right-hand sides (as positive numbers).
- *Spiking vector*: defines a selection of rules to be fired in a transition step, using a position per rule. Given the non-deterministic nature of SNP systems, there are more than one valid spiking vectors for a given configuration.
- *Configuration vector*: defines the number of spikes per neuron, that is, the configuration in a given time.

The algorithm is then used to compute the next configuration from a given one by performing only vector-matrix operations. These operations are offloaded to the GPU, which is optimized to handle matrices [15]. CuSNP is a simulator for SNP systems that is written in Python and the CUDA kernels were launched by using the binding library PyCUDA. For the first approach, SNP systems without delays were simulated by covering each computation path sequentially, leading to speedups of up to 2.31× [15, 61].

Further extensions have followed, enabling delays, support of more types of regular expressions, and input of P-Lingua files [17]. The support of delays is done by an extension of the matrix representation, introducing vectors to control when rules fire after the delay is met and if neurons are open. These led to up to 50× of acceleration on a GTX750 GPU [17]. Moreover, non-deterministic SNP systems with delays are supported by generating all computation paths sequentially and using GPU to speed up the simulation [16] (with up to 2× on a GTX1070 for a non-uniform solution to Subset Sum).

Furthermore, the simulation of Fuzzy Reasoning Spiking Neural (FRSN) P systems on the GPU was explored [55]. FRSNP systems allows modeling fuzzy diagnosis knowledge and reasoning for fault diagnosis applications. The simulation algorithm is also based on a matrix representation and vector-matrix operations. The employed simulation framework was pLinguaCore, so the CUDA kernels were launched by using the binding library JCUDA.

7.4 Other Models

Enzymatic Numerical P systems has been employed for modeling robot controllers and path planning, being significant for the Artificial Intelligence. A CUDA simulator was developed [61], where the selection of applicable programs was first applied; second, the calculation of production functions; and third, distribution of production function results according to repartition protocols. Production functions are computed using a recursive solution. Simulators were implemented in Java (inside pLinguaCore) and C programming languages as standalone tools. On a GeForce GTX 460M, the achieved speedup was of up to 11×. Moreover, a model for RRT and RRT* algorithms for robotic motion planning using ENPS was also simulated on CUDA [79] 24× faster than in multicore CPU using an RTX2080.

Table 6. The Summing up of CUDA-GPU Implementations I

Item	PCUDA	PCUDASAT	TSPCUDASAT
P system class	Cell-like (recognizer P system with active membranes).	Cell-like P system with active membranes (resolve SAT in linear time).	Tissue-like P system (resolve SAT in linear time)
P system ingredients representation	Elementary membranes are represented by thread blocks. Threads are assigned to objects.	Each elementary is assigned a thread block. Only the objects appearing in the input multisets will be assigned a thread.	Every cell is assigned a thread block. More objects are requested by every cell than the PCUDASAT simulator.
Operation Process	A transition step is performed in selection stage and execution stage.	1. Generation; 2. Synchronization; 3. Check-out; 4. Output.	1. Generation; 2. Exchange; 3. Synchronization; 4. Check-in; 5. Output.
Experiment models	(a) An illustrator model to stress the simulator. (b) A model aims at solving a SAT problem.	A sequential and two CUDA-based parallel simulators (one of them is designed by a hybrid method).	A CUDA-based simulator.
Size of the simulated P system	512 objects and 1,024 membranes for (a) and 914 objects and 4,096 membranes for (b).	256 objects and 4,096 membranes.	256 objects and 4,096 membranes.
Speedup	7× for (a) and 1.67× for (b).	The CUDA simulator is 63× faster than the sequential simulator and the hybrid CUDA simulator is 9.3× faster than the normal one.	10× higher than the sequential simulator.

The target CUDA-GPU is NVIDIA C1060 (240 cores) and the host platform is a computer with two Intel i5 Nehalem processors (8 cores).

Evolution-Communication P systems with Energy (ECPE) [61] were also target for a CUDA simulation. The employed simulation algorithm used a matrix representation and linear-algebra–based algorithm, similarly as for the spiking neural P systems simulator: a configuration vector, a trigger matrix, an application vector, and a transition vector.

The summary of CUDA-GPU implementations is concluded in Tables 6 and 7, providing an easy checking for the CUDA-based simulations for academic communities. For the sake of presenting a overview contrast of FPGA and CUDA-GPU implementation, we summarize the quantitative and qualitative attributes and conclude their methodologies in Tables 8, 9, and 10. This overall contrast presents a recapitalized conclusion about hardware implementations of P system.

8 OTHER APPROACHES

Besides FPGA-based and CUDA-enabled GPU-based hardware platforms developed for the implementation of P systems, there are several attempts to implement P systems on micro-controllers. Serial algorithms and their hardware circuit designs in terms of the *exhaustive investigation line* focusing on implementing the transition of configurations of P systems were developed. The corresponding research does not target concrete hardware devices, it is just designing the circuits aiming at simulating certain operations of particular P systems with registers, logical gates, magnitude comparators, and data buses.

In Reference [32], a digital circuit is presented to select *active* rules in the current configuration. Each evolution rule is represented by two hardware registers. The first register characterizes the left-hand side (antecedent) of a rule and the other specifies the right-hand side of the rule, which also determines whether the rewritten objects go out from the current membrane or stay where they are, or go into inner membranes. By comparing the left-hand side of each rule with the multiset of objects in a region, the applicability of every rule can be determined. Next, an

Table 7. The Summing up of CUDA-GPU Implementations II

Item	ABCD-GPU	ENPS-GPU	snpgpu and CuSNP
P system class	Population Dynamics P system.	Enzymatic numerical P system (cell-like).	Spiking Neural P system.
P system ingredients representation	Environments and simulations are distributed through thread blocks. Rule blocks are assigned among threads.	Each production function and each repartition protocol element is associated with a thread, respectively.	Spiking transition matrices, spiking vectors, and configuration vectors are employed to model the target systems.
Operation process	(DCBA algorithm) Selection stage (1. Distribution; 2. Maximality; 3. Probability) and Execution stage.	1. Select the applicable programs; 2. Compute production functions; 3. Distribute the results of the precede step according to repartition protocols.	Generate the matrix representation for the model and iterate the computation of next configuration until a stopping criterion is met.
Experiment models	A set of randomly generated PDP systems (without biological meaning), the ecosystem model of the Bearded Vulture and a tritrophic example model with modules defined as features.	A dummy model and an exponential function approximation model.	(a) SNP systems covering each computation path sequentially, without delays. (b) Extensions enabling delays, support of more types of regular expressions and input of P-Lingua files.
Size of the simulated P system	Running 50 simulations, 20 environments, and more than 20,000 rule blocks.	Two membranes, range from 1 to 120,000 programs.	(a) From 1 to 16 neurons. (b) Generalized sorting networks with up to 512 neurons.
Speedup	7× (Tesla C1060) compared with the sequential model, 3× contrasted to the four cores CPU. The speedup for the real ecosystem, 4.9× with a C1060 and 18.1× using a Tesla K40 GPU (2,880 cores). Using the adaptative simulator with the tritrophic model using modules achieved an extra 2.5× on a K40 GPU and 1.8× on a P100 GPU (3,584 cores).	6.5× for dummy model and 10× for exponential model. 11× on GeForce GTX 460M.	(a) 2.31× for the Python version simulator. (b) 50× on CUDA-based version (GTX750 GPU).

algorithm computes the number of applications of active rules given the multiset of objects and evolution rules [58]. The corresponding circuit is composed of logical gates, registers, multiplexers, and sequential elements. The computation process is bounded. In Reference [47], a P system circuit is constructed by means of PIC16F88 micro-processor enriched with the storage component 24LC1025, connected by an I2C bus. Thus, the shortcoming of insufficient storage capacity of the micro-processor is solved by the introduced external memory. The flexibility of the circuit is acceptable, as the modification of the circuits is not difficult.

Continuing the research from Reference [58], Reference [59] presents an improved algorithm and the corresponding circuit calculating the application times of active rules. The computing process can be completed in minor steps and the theoretical performance is optimized. In References [6, 7], a draft of a circuit implementing the inherent parallelism of P system is presented. Towards the parallelism, the rule treatment is similar with the *region-based solution* defined in Section 6, namely, what applied is a multiset of rules instead of a single rule. An operating environment is elaborated in Reference [42], which performs the automatic transformation of tasks involved in the hardware simulation of P systems, including loading, execution, and interpretation, into a distributed framework constructed on micro-controllers. The execution results of the circuit, which are in the form of binary data, can be interpreted in a transparent manner.

Table 8. The Comparison of the Quantitative Attributes of the Implementations between FPGA and CUDA-GPU

Item	FPGA	CUDA-GPU
Inchoate year	2003	2009
Hardware Price	\$50–\$4500	\$150–\$2000.
Concerned institute	Southwest Jiaotong University, Université Paris-Est Créteil Val de Marne, Xihua University	Universidad de Sevilla, University of the Philippines Diliman
Number of researchers	6	8
Implemented P system type	Cell-like P system	P systems with active membranes, tissue P systems with cell division for a SAT solution, Population dynamics P system, spiking Neural P system, enzymatic Numerical P system, evolution communication P system.
Size of the simulated P systems	No limit for objects and membranes, but less than 1,000 rules	No limit as long as GPU resources (mainly memory) are available.
Approximate time needed to obtain a hardware system	Two months	One month.
Speedup effect	2×10^7 computational steps per second at present.	Depends on model, implementation, or GPU device. Reported speedups from 2× to 90× compared to a sequential counterpart.
Handling of non-determinism	DND algorithm and formal power serial theory.	DCBA, or ad hoc strategies (e.g., assuming confluent P systems).

What should be emphasized here is that all the hardware circuits introduced are just on the blueprints, which were never implemented in practice. They remain mostly theoretical and the actual functionality and performances are unknown, unlike for FPGA-based and CUDA-based hardware implementations/simulations that are carried out practically. In Reference [45], microcontrollers are also chosen as target hardware to implement communication architectures of P systems. A digital circuit carrying out massive parallelism in transition P Systems is established in Reference [6].

As a new attempt for implementing neural P systems on different h ardware, DRAM-based CMOS circuits are adopted to construct elementary spiking neural P systems. We do not carry out an in-depth discussion about this topic and refer to Reference [113] for more details.

9 CONCLUSIONS

In this article, we gave an overview of different existing hardware implementations of P systems. As expected, the FPGA-based approach is very promising, yielding truly parallel implementations achieving a speedup of order 10^5. However, it requires a considerable amount of work, as the corresponding hardware architecture should be created from scratch. Moreover, the hardware description languages like VHDL are very low-level and programming in such languages tends to be extremely tedious. Also, as it was shown from the existing implementations, the most promising results are obtained using rule-based architectures, because of the smaller communication cost.

By contrast, CUDA-based approach features a powerful unique hardware platform that can be programmed using standard C/C++ language. While it gives less freedom and accuracy, this approach gives a good trade-off between complexity, scalability, and maintenance. The achieved speedups are relatively small (between 4× and 50×), however, due to the programming simplicity, more types of P systems were implemented.

Table 9. The Comparison of Qualitative Attributes of the Implementations between FPGA and CUDA-GPU

Item	FPGA	CUDA-GPU
Complexity of the parallel framework	Both region-level and system-level parallelism are realized, as well as the non-determinism (the equiprobability of the rules are guaranteed in Juan's contribution).	Both region-level and system-level parallelism are realized, several strategies are carried out to sort out non-determinism (for PDP systems, based on DND algorithm in which the equiprobability is not proven mathematically).
Extensibility	Although the hardware consumptions are no more than 30% when rules are no more than 300, which means there are amount of space to accommodate new features of the P system, extending the existing type of P system to additional types needs significant revisions.	The programming framework is flexible enough to support other ingredients, but it might require a significant revision to the parallel design.
Scalability	With the size of the subject P systems growing, the computation clock rates decrease not too much and the hardware usages are approximately proportional to the size.	Newer GPUs showed to reach higher performance, but the simulations have been demonstrated to be memory bandwidth bounded. It requires to run large models to see a positive speedup.
Experiment subject	Cell-like P systems with 10–1,000 rules, 1 to 55 regions, 3–165 objects.	(a) P systems with active membranes with 914 objects/rules and 4,096 membranes, (b) SAT solutions in active membranes and cell division 256 objects/rules and 4,096 membranes, (c) PDP systems up to 200,000 rules on 50 simulations and 20 environments, (d) SN P systems with 16 to 512 neurons, (e) ENP systems with 2 membranes and up to 120,000 programs.
Difficulties	Programming with HDL is difficult contrasting with its software counterpart, and the synchronization of intricate circuits. It usually costs 8–10 months to be proficient with HDL programming and to comprehend the parallel architecture.	Work assignation for a parallel design can be the most difficult task when developing GPU simulators, since the restrictions of GPU computing meets with P system semantics. Need to think on GPU architecture for high performance: memory accesses, memory transfers, thread synchronization, and so on.
Application	The prototypes are developed but no applications for concrete problems to date.	Several applications for biological simulations have to executed. Solutions to computationally hard problems.

The central problem in both approaches—the object distribution problem—was tackled from different points of view, the most fruitful attempts being the variations of the DND algorithm and direct approaches using mathematical properties of the dependency relation between rules. Future research can be done in this direction by providing new classes of P systems suitable for the precomputation of possible rule applications. Another research direction related to software implementation of P systems is given by the construction from Section 4, which features a novel reduction of the object distribution problem to ILP. This reduction handles well the maximal parallelism and allows to define criteria for the choice of the solution. Then, it would be possible to design a software framework that would use existing solvers and algorithms to quickly obtain the desired solution, including those optimized for parallel hardware [10, 23, 96, 97, 102, 120].

Finally, as a future research interest, we mention a robust implementation of different features of P systems like membrane creation/dissolution, as well as non-classical variants of P systems like numerical P systems. The motivation for such research is that the corresponding models feature numerous applications, so their scalable implementation would immediately allow to test the acceleration of corresponding algorithms and their practical usage.

Table 10. The Conclusion of the Implementations for FPGA and CUDA-GPU

Item	FPGA	CUDA-GPU
Advantages	The computation speed is fast, and the hardware framework is adjustable. The compromises between implementing the P system model and constructing the most proximate hardware are relatively ideal.	Relatively convenient way to implement P systems for the established parallel framework in the GPU. The programming is similar with the classic software developing process.
Disadvantages	Building the parallel architecture from scratch is a laborious and challenging venture.	Slower than FPGA. Sometimes big concessions are indispensable, as the architectures are unchangeable.
Contributions	Introducing the reconfigurable hardware to develop the real parallel architectures that can exploit the maximal parallelism of the P systems substantially.	The tremendous potential of the emergent universal computing GPU is concentrated, which the ready-made parallel framework is off-the-shelf, to establish a P system computing platform conveniently.
Conclusions	We can develop sophisticated P systems hardware circuits that implement the target models as approximate as possible on FPGA devices. The expense we should pay is also considerablly high, taking the painstaking effort into account.	CUDA-GPU provides a relatively comfortable and alternative choice to implement P systems, which the simulation results are acceptable, nevertheless at a price of the modification of the source models. GPUs worth when simulating large P system models.

ACKNOWLEDGMENTS

The authors are grateful to the Editor-in-Chief, Prof. Sartaj Sahni, the anonymous handling editor and all reviewers for their insightful and detailed comments on this manuscript, and are also indebted to Academician Gheorghe Păun for his useful discussions and valuable suggestions.

REFERENCES

[1] Rick Merritt. 2017. Roadmap Says CMOS Ends ~2024. IRDS points to chip stacks, new architectures. Retrieved from https://web.archive.org/web/20170324022546/, https://www.eetimes.com/document.asp?doc_id=1331517.

[2] Gordon Moore. 1975. Progress in digital integrated electronics. *International Electron Devices Meeting*. IEEE, 11–13.

[3] Oana Agrigoroaiei, Gabriel Ciobanu, and Andreas Resios. 2010. Evolving by maximizing the number of rules: Complexity study. In *Proceedings of the 10th International Workshop on Membrane Computing (WMC'09) (LNCS)*, Gheorghe Păun, Mario J. Pérez-Jiménez, Agustín Riscos-Núñez, Grzegorz Rozenberg, and Arto Salomaa (Eds.). Springer, 149–157.

[4] Bruce Alberts, Alexander Johnson, Julian Lewis, Martin Raff, Keith Roberts, and Peter Walter. 2002. *Molecular Biology of the Cell* (4th ed.). Garland.

[5] Artiom Alhazov. 2005. Maximally parallel multiset-rewriting systems: Browsing the configurations. In *Proceedings of the 3rd Brainstorming Week on Membrane Computing (RGNC Report)*, M. A. Gutiérrez-Naranjo, A. Riscos-Núñez, F. J. Romero-Campero, and D. Sburlan (Eds.). 1–10.

[6] Santiago Alonso, Luis Fernández, Fernando Arroyo, and Javier Gil. 2008. A circuit implementing massive parallelism in transition P systems. *Int. J. Inform. Technol. Knowl.* 2, 1 (2008), 35–42.

[7] Santiago Alonso, Luis Fernández, Fernando Arroyo, and Javier Gil. 2008. Main modules design for a HW implementation of massive parallelism in transition P-systems. *Artif. Life Robot.* 13, 1 (2008), 107–111.

[8] Himanshu Kushwah Anil Sethi. 2015. Multicore processor technology—Advantages and challenges. *Int. J. Res. Eng. Technol.* 4, 9 (2015), 87–89.

[9] Alberto Arteta, Luis Fernández, and Javier Gil. 2008. Algorithm for application of evolution rules based on linear diofantic equations. In *Proceedings of the 10th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC'08)*, Viorel Negru, Tudor Jebelean, Dana Petcu, and Daniela Zaharie (Eds.). IEEE Computer Society, 496–500.

[10] Jambhlekar P. Arun, Manoj Mishra, and Sheshasayee V. Subramaniam. 2011. Parallel implementation of MOPSO on GPU using OpenCL and CUDA. In *Proceedings of the 18th International Conference on High Performance Computing*. 1–10.

[11] Angel V. Baranda, Fernando Arroyo, Juan Castellanos, and Rafael Gonzalo. 2001. Towards an electronic implementation of membrane computing: A formal description of non-deterministic evolution in transition P systems. In

Proceedings of the 7th International Workshop on DNA-Based Computers: DNA Computing (LNCS), N. Jonoska and N. C. Seeman (Eds.), Vol. 2340. Springer, 350–359.

[12] Sankar Basu, Randal E. Bryant, Giovanni De Micheli, Thomas Theis, and Lloyd Whitman. 2019. iFLEX: A fully open-source high-density field-programmable gate array (FPGA)-based hardware co-processor for vector similarity searching. *Proc. IEEE* 107, 1 (2019), 11–18.

[13] Francesco Bernardini and Marian Gheorghe. 2004. Population P systems. *J. Univ. Comput. Sci.* 10, 5 (2004), 509–539.

[14] Catalin Buiu, Cristian Vasile, and Octavian Arsene. 2012. Development of membrane controllers for mobile robots. *Inf. Sci.* 187 (2012), 33–51.

[15] Francis G. C. Cabarle, Henry N. Adorna, Miguel A. Martínez-del-Amor, and Mario J. Pérez-Jiménez. 2012. Improving GPU simulations of spiking neural P systems. *Roman. J. Inf. Sci. Technol.* 15, 1 (2012), 5–20.

[16] Jym P. Carandang, Francis G. C. Cabarle, Henry N. Adorna, Nestine H. S. Hernandez, and Miguel Á. Martínez-del-Amor. 2019. Handling non-determinism in spiking neural P systems: Algorithms and simulations. *Fundam. Inform.* 164 (2019), 139–155.

[17] Jym P. A. Carandang, John M. B. Villaflores, Francis G. C. Cabarle, Henry N. Adorna, and Miguel A. Martínez-del-Amor. 2017. CuSNP: Spiking neural P systems simulators in CUDA. *Roman. J. Inform. Sci. Technol.* 20, 1 (2017), 57–70.

[18] José M. Cecilia, José M. García, Ginés D. Guerrero, Miguel A. Martínez-del-Amor, Ignacio Pérez-Hurtado, and Mario J. Pérez-Jiménez. 2010. Simulation of P systems with active membranes on CUDA. *Brief. Bioinform.* 11, 3 (2010), 313–322.

[19] José M. Cecilia, José M. García, Ginés D. Guerrero, Miguel A. Martínez-del-Amor, Mario J. Pérez-Jiménez, and Manuel Ujaldon. 2012. The GPU on the simulation of cellular computing models. *Soft Comput.* 16, 2 (2012), 231–246.

[20] Gabriel Ciobanu, Solomon Marcus, and Gheorghe Păun. 2009. New strategies of using the rules of a P system in a maximal way: Power and complexity. *Roman. J. Inform. Sci. Technol.* 12, 2 (2009), 157–173.

[21] Gabriel Ciobanu and Andreas Resios. 2009. Complexity of evolution in maximum cooperative P systems. *Nat. Comput.* 8, 4 (31 Jan. 2009), 807.

[22] Gabriel Ciobanu and Guo Wenyuan. 2003. P systems running on a cluster of computers. In *Proceedings of the International Workshop Membrane Computing (WMC'03) (Lecture Notes in Computer Science)*, Carlos Martín-Vide, Giancarlo Mauri, Gheorghe Páun, Grzegorz Rozenberg, and Arto Salomaa (Eds.), Vol. 2933. Springer, 123–139.

[23] Daniele D'Agostino, Giulia Pasquale, and Ivan Merelli. 2014. A fine-grained CUDA implementation of the multi-objective evolutionary approach NSGA-II: Potential impact for computational and systems biology applications. In *Proceedings of the 11th International Meeting of Computational Intelligence Methods for Bioinformatics and Biostatistics (CIBB'14) (LNCS)*, Clelia Di Serio, Pietro Liò, Alessandro Nonis, and Roberto Tagliaferri (Eds.), Vol. 8623. Springer, 273–284.

[24] Ren T. A. de la Cruz, Francis G. C. Cabarle, and Henry N. Adorna. 2019. Generating context-free languages using spiking neural P systems with structural plasticity. *J. Memb. Comput.* 1, 3 (2019), 161–177.

[25] Daniel Díaz-Pernil, Miguel A. Gutiérrez-Naranjo, and Hong Peng. 2019. Membrane computing and image processing: A short survey. *J. Memb. Comput.* (04 Feb. 2019).

[26] Matthew S. Dodd, Dominic Papineau, Tor Grenne, John F. Slack, Martin Rittner, Franco Pirajno, Jonathan O'Neil, and Crispin T. S. Little. 2017. Evidence for early life in Earth's oldest hydrothermal vent precipitates. *Nature* 543, 7643 (2017), 60–64.

[27] Matthias Ehrgott. 2005. *Multicriteria Optimization* (2nd ed.). Springer.

[28] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger. 2012. Neural acceleration for general-purpose approximate programs. In *Proceedings of the 45th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'12)*. IEEE Computer Society, 449–460.

[29] R. Uribe, F. Varela, and H. Maturana. 1974. Autopoiesis: The organization of living systems, its characterization and a model. *BioSystems* 5, 4 (1974), 187–196.

[30] Luis Fernández, Fernando Arroyo, Ivan Garcia, and Gines Bravo. 2007. Decision trees for applicability of evolution rules in transition P systems. *Inf. Theor. Applic.* 14, 3 (2007), 223–230.

[31] Luis Fernández, Fernando Arroyo, Jorge A. Tejedo, and Juan Castellanos. 2006. Massively parallel algorithm for evolution rules application in transition P systems. In *Proceedings of the 7th Workshop on Membrane Computing*, Hendrik Jan Hoogeboom, Gheorghe Păun, and Grzegorz Rozenberg (Eds.). Universiteit Leiden, 337–343.

[32] Luis Fernández, Victor J. Martínez, Fernando Arroyo, and Luis F. Mingo. 2005. A hardware circuit for selecting active rules in transition P systems. In *Proceedings of the 7th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC'05)*, Daniela Zaharie, Dana Petcu, Viorel Negru, Tudor Jebelean, Gabriel Ciobanu, Alexandru Cicortas, Ajith Abraham, and Marcin Paprzycki (Eds.). IEEE Computer Society, 415–418.

[33] Rudolf Freund, Alberto Leporati, Giancarlo Mauri, Antonio E. Porreca, Sergey Verlan, and Claudio Zandron. 2013. Flattening in (Tissue) P systems. In *Proceedings of the 14th International Conference on Membrane Computing*

(CMC'13) (Lecture Notes in Computer Science), Artiom Alhazov, Svetlana Cojocaru, Marian Gheorghe, Yurii Rogozhin, Grzegorz Rozenberg, and Arto Salomaa (Eds.), Vol. 8340. Springer, 173–188.

[34] Rudolf Freund, Ignacio Pérez-Hurtado, Agustín Riscos-Núñez, and Sergey Verlan. 2013. A formalization of membrane systems with dynamically evolving structures. *Int. J. Comput. Math.* 90, 4 (2013), 801–815.

[35] Rudolf Freund and Sergey Verlan. 2007. A formal framework for static (Tissue) P systems. In *Proceedings of the 8th International Workshop on Membrane Computing (WMC'07) (Lecture Notes in Computer Science)*, George Eleftherakis, Petros Kefalas, Gheorghe Paun, Grzegorz Rozenberg, and Arto Salomaa (Eds.), Vol. 4860. Springer, 271–284.

[36] Zsolt Gazdag and Gábor Kolonits. 2019. A new method to simulate restricted variants of polarizationless P systems with active membranes. *J. Memb. Comput.* 1, 4 (2019), 251–261.

[37] Marian Gheorghe, Andrei Păun, Sergey Verlan, and Gexiang Zhang. 2017. *Membrane Computing, Power and Complexity*. Springer Berlin, 1–16.

[38] Francisco J. Gil, Luis Fernández, Fernando Arroyo, and Juan Alberto de Frutos. 2008. Parallel algorithm for P systems implementation in multiprocessors. In *Proceedings of the 13th International Symposium on Artificial Life and Robotics (AROB'08)*, M. Sugisaka and H. Tanaka (Eds.). 10–25.

[39] Francisco J. Gil, Luis Fernández, Fernando Arroyo, and Jorge A. Tejedor. 2007. Delimited massively parallel algorithm based on rules elimination for application of active rules in transition P systems. In *Proceedings of the 5th International Conference on Information Research and Applications (i.TECH'07)*, Krassimir Markov and Krassimira Ivanova (Eds.), Vol. 1. Institute of Information Theories and Applications FOI ITHEA, Bulgaria, 182–188.

[40] Francisco J. Gil, Jorge A. Tejedor, and Luis Fernández. 2008. Fast linear algorithm for active rules application in transition P systems. In *Algorithmic and Mathematical Foundations of the Artificial Intelligence (International Book Series INFORMATION SCIENCE & COMPUTING)*, Krassimir Markov, Krassimira Ivanova, and Ilia Mitov (Eds.), Vol. Supple. Institute of Information Theories and Applications FOI ITHEA, Sofia, Bulgaria, 35–44.

[41] Fernando Arroyo Ginés Bravo, Luis Fernández, and Juan Frutos. 2008. A hierarchical architecture with parallel communication for implementing P systems. *Inf. Technol. Knowl.* 2, 1 (2008), 43–48.

[42] Sandra M. Gomez-Canaval, Abraham Gutiérrez, and Santiago Alonso. 2008. Hardware implementation of P systems using microcontrollers. An operating environment for implementing a partially parallel distributed architecture. In *Proceedings of the 10th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC'08)*, Viorel Negru, Tudor Jebelean, and Dana Petcuand Daniela Zaharie (Eds.). IEEE, 489–495.

[43] Ananth Grama, George Karypis, Vipin Kumar, and Anshul Gupta. 2003. *Introduction to Parallel Computing* (2nd ed.). Addison-Wesley.

[44] Abraham Gutiérrez, Luis Fernández, Fernando Arroyo, and Santiago Alonso. 2007. Hardware and software architecture for implementing membrane systems: A case of study to transition P systems. In *Proceedings of the 13th International Meeting on DNA Computing (LNCS)*, Max H. Garzon and Hao Yan (Eds.), Vol. 4848. Springer, 211–220.

[45] Abraham Gutiérrez, Luis Fernández, Fernando Arroyo, and Santiago Alonso. 2008. Suitability of using microcontrollers in implementing new P-system communications architectures. *Artif. Life Robot.* 13, 1 (01 Dec. 2008), 102–106.

[46] Abraham Gutiérrez, Luis Fernández, Fernando Arroyo, and Ginés Bravo. 2007. Optimizing membrane system implementation with multisets and evolution rules compression. In *Proceedings of the 8th Workshop on Membrane Computing*, G. Ekeftherakis, P. Kefalas, and Gh. Păun (Eds.). 345–362.

[47] Abraham Gutiérrez, Luis Fernández, Fernando Arroyo, and Victor J. Martínez. 2006. Design of a hardware architecture based on microcontrollers for the implementation of membrane systems. In *Proceedings of the 8th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, Viorel Negru, Dana Petcu, Daniela Zaharie, Ajith Abraham, Bruno Buchberger, Alexandru Cicortas, Dorian Gorgan, and Joël Quinqueton (Eds.). IEEE Computer Society, 350–353.

[48] Francisco Javier Gil, Luis Fernández, Fernando Arroyo, and Jorge Tejedor. 2008. Delimited massively parallel algorithm based on rules elimination for application of active rules in transition P systems. *Inf. Technol. Knowl.* 2, 1 (2008), 56–61.

[49] Z. B. Jimenez, Francis G. C. Cabarle, Ren T. A. de la Cruz, Kelvin C. Buño, Henry N. Adorna, Nestine H. S. Hernandez, and Xiangxiang Zeng. 2019. Matrix representation and simulation algorithm of spiking neural P systems with structural plasticity. *J. Memb. Comput.* 1, 3 (2019), 145–160.

[50] Ignacy Kaliszewski, Janusz Miroforidis, and Dmitry Podkopaev. 2016. *Multiple Criteria Decision Making by Multiobjective Optimization–A Toolbox*. Int. Series in Operations Research & Management Science, Vol. 242. Springer.

[51] Takahiro Katagiri. 2019. High-performance computing basics. In *The Art of High Performance Computing for Computational Science, Vol. 1, Techniques of Speedup and Parallelization for General Purposes*, Masaaki Geshi (Ed.). Springer, 1–25.

[52] David Kirk and Wen-Mei Hwu. 2010. *Programming Massively Parallel Processors: A Hands On Approach*. Morgan Kaufmann.

[53] Julien Legriel. 2011. *Multi-Criteria Optimization and Its Application to Multi-Processor Embedded Systems*. Universite de Grenoble, Grenoble, France.

[54] Jianping Kelvin Li, Jia-Kai Chou, and Kwan-Liu Ma. 2015. High performance heterogeneous computing for collaborative visual analysis. In *Proceedings of the SIGGRAPH Asia Visualization in High Performance Computing Conference*. ACM, 12:1–12:4.

[55] Luis F. Macías-Ramos, Miguel A. Martínez-del-Amor, and Mario J. Pérez-Jiménez. 2015. Simulating FRSN P systems with real numbers in P-Lingua on sequential and CUDA platforms. In *Proceedings of the 16th International Conference on Membrane Computing (CMC'15) (LNCS)*, G. Rozenberg, A. Salomaa, J. M. Sempere, and C. Zandron (Eds.). 262–276.

[56] Vincenzo Manca. 2019. From biopolymer duplication to membrane duplication and beyond. *J. Memb. Comput.* 1, 4 (2019), 292–303.

[57] Víctor Martínez, Santiago Alonso, and Abraham Gutiérrez. 2010. Hardware circuit for the application of evolution rules in a transition P-system. *Artif. Life Robot.* 15, 1 (01 Aug. 2010), 89–92.

[58] Victor Martínez, Fernando Arroyo, Abraham Gutiérrez, and Luis Fernández. 2006. Hardware implementation of a bounded algorithm for application of rules in a transition P-system. In *Proceedings of the 8th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC'06)*, Viorel Negru, Dana Petcu, Daniela Zaharie, Ajith Abraham, Bruno Buchberger, Alexandru Cicortas, Dorian Gorgan, and Joel Quinqueton (Eds.). IEEE, 343–349.

[59] Victor Martínez, Luis Fernández, Fernando Arroyo, and Abraham Gutiérrez. 2007. HW implementation of a optimized algorithm for the application of active rules in a transition P-system. *Inf. Theor. Applic.* 14, 4 (2007), 324–331.

[60] Victor J. Martínez, Fernando Arroyo, Abraham Gutiérrez, and Luis Fernández. 2006. Hardware implementation of a bounded algorithm for application of rules in a transition P-system. In *Proceedings of the 8th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC'06)*, Viorel Negru, Dana Petcu, Daniela Zaharie, Ajith Abraham, Bruno Buchberger, Alexandru Cicortas, Dorian Gorgan, and Joël Quinqueton (Eds.). IEEE Computer Society, 343–349.

[61] Miguel A. Martínez-del-Amor, Manuel García-Quismondo, Luis F. Macías-Ramos, Luis Valencia-Cabrera, Agustin Riscos-Núñez, and Mario J. Pérez-Jiménez. 2015. Simulating P systems on GPU devices: A survey. *Fundam. Inform.* 136, 3 (2015), 269–284.

[62] Miguel A. Martínez-del-Amor, Luis F. Macías-Ramos, Luis Valencia-Cabrera, and Mario J. Pérez-Jiménez. 2016. Parallel simulation of population dynamics P systems: Updates and roadmap. *Nat. Comput.* 15, 4 (2016), 565–573.

[63] Miguel A. Martínez-del-Amor, Ignacio Pérez-Hurtado, Manuel García-Quismondo, Luis F. Macías-Ramos, Luis Valencia-Cabrera, Álvaro Romero Jiménez, Carmen Graciani Díaz, Agustin Riscos-Núñez, Maria Angels Colomer, and Mario J. Pérez-Jiménez. 2012. DCBA: Simulating population dynamics P systems with proportional object distribution. In *Proceedings of the 13th International Conference on Membrane Computing (CMC'12) (Lecture Notes in Computer Science)*, Erzsébet Csuhaj-Varjú, Marian Gheorghe, Grzegorz Rozenberg, Arto Salomaa, and György Vaszil (Eds.). 257–276.

[64] Miguel Á. Martínez-del-Amor, Ignacio Pérez-Hurtado, David Orellana-Martín, and Mario J. Pérez-Jiménez. 2020. Adaptative parallel simulators for bioinspired computing models. *Fut. Gen. Comput. Syst.* 107 (2020), 469–484.

[65] Miguel A. Martínez-del-Amor, Ignacio Pérez-Hurtado, Mario J. Pérez-Jiménez, Agustin Riscos-Núñez, and M. Angels Colomer. 2010. A new simulation algorithm for multienvironment probabilistic P systems. In *Proceedings of the 5th International Conference on Bio-Inspired Computing: Theories and Applications*, M. Gong, L. Pan, T. Song, and G. Zhang (Eds.). 59–68.

[66] Neil Mathur. 2002. Beyond the silicon roadmap. *Nature* 419 (Oct. 2002), 573–575.

[67] George H. Mealy. 1955. A method for synthesizing sequential circuits. *Bell Syst. Tech. J.* 34, 5 (1955), 1045–1079.

[68] Anthony Nash and Sara Kalvala. 2019. A P system model of swarming and aggregation in a Myxobacterial colony. *J. Memb. Comput.* 1, 2 (2019), 103–111.

[69] Van Nguyen. 2010. *An Implementation of the Parallelism, Distribution and Nondeterminism of Membrane Computing Models on Reconfigurable Hardware*. Ph.D. Dissertation. University of South Australia.

[70] Van Nguyen, David Kearney, and Gianpaolo Gioiosa. 2007. Balancing performance, flexibility, and scalability in a parallel computing platform for membrane computing applications. In *Proceedings of the 8th International Workshop on Membrane Computing (LNCS)*, G. Eleftherakis, P. Kefalas, Gh. Păun, G. Rozenberg, and A. Salomaa (Eds.), Vol. 4860. Springer, 385–413.

[71] Van Nguyen, David Kearney, and Gianpaolo Gioiosa. 2008. An algorithm for non-deterministic object distribution in P systems and its implementation in hardware. In *Proceedings of the 9th International Workshop on Membrane Computing (WMC'08) (LNCS)*, D. W. Corne, P. Frisco, Gh. Păun, G. Rozenberg, and A. Salomaa (Eds.), Vol. 5391. Springer, 325–354.

[72] Van Nguyen, David Kearney, and Gianpaolo Gioiosa. 2008. An implementation of membrane computing using reconfigurable hardware. *Comput. Inform.* 27, 3 (2008), 551–569.

[73] Van Nguyen, David Kearney, and Gianpaolo Gioiosa. 2009. A region-oriented hardware implementation for membrane computing applications. In *Proceedings of the 10th International Workshop on Membrane Computing (WMC'09) (LNCS)*, Gh. Păun, M. J. Pérez-Jiménez, A. Riscos-Núñez, G. Rozenberg, and A. Salomaa (Eds.), Vol. 5957. Springer, 385–409.

[74] Van Nguyen, David Kearney, and Gianpaolo Gioiosa. 2010. An extensible, maintainable and elegant approach to hardware source code generation in Reconfig-P. *J. Logic Algeb. Program.* 79, 6 (2010), 383–396.

[75] Taishin Y. Nishida. 2006. *A Membrane Computing Model of Photosynthesis*. Springer, 181–202.

[76] David Orellana-Martín, Miguel A. Martínez-del-Amor, Luis Valencia-Cabrera, Bosheng Song, Linqiang Pan, and Mario J. Pérez-Jiménez. 2020. P systems with symport/antiport rules: When do the surroundings matter? *Theoret. Comput. Sci.* 805 (2020), 206–217.

[77] David Orellana-Martín, Luis Valencia-Cabrera, Agustín Riscos-Núñez, and Mario J. Pérez-Jiménez. 2019. Minimal cooperation as a way to achieve the efficiency in cell-like membrane systems. *J. Memb. Comput.* 1, 2 (2019), 85–92.

[78] Ana Pavel, Octavian Arsene, and Catalin Buiu. 2010. Enzymatic numerical P systems—A new class of membrane computing systems. In *Proceedings of the 5th International Conference on Bio-Inspired Computing: Theories and Applications (BIC-TA'10)*, A. K. Nagar, R. Thamburaj, K. Li, Z. Tang, and R. Li (Eds.). IEEE, 1331–1336.

[79] Ignacio Pérez-Hurtado, Miguel Á. Martínez-del-Amor, Gexiang Zhang, Ferrante Neri, and Mario J. Pérez-Jiménez. 2020. A membrane parallel rapidly-exploring random tree algorithm for robotic motion planning. *Integ. Comput.-Aided Eng.* 27 (2020), 121–138.

[80] Ignacio Pérez-Hurtado, David Orellana-Martín, Gexiang Zhang, and Mario J. Pérez-Jiménez. 2019. P-Lingua in two steps: Flexibility and efficiency. *J. Memb. Comput.* 1, 2 (01 June 2019), 93–102.

[81] Ignacio Pérez-Hurtado, Mario J. Pérez-Jiménez, Gexiang Zhang, and David Orellana-Martín. 2018. Simulation of rapidly-exploring random trees in membrane computing with P-lingua and automatic programming. *Int. J. Comput. Commun. Contr.* 13, 6 (2018), 1007–1031.

[82] Biljana Petreska and Christof Teuscher. 2003. A reconfigurable hardware membrane system. In *Proceedings of the International Workshop on Membrane Computing (WMC'03) (Lecture Notes in Computer Science)*, Carlos Martín-Vide, Giancarlo Mauri, Gheorghe Paun, Grzegorz Rozenberg, and Arto Salomaa (Eds.), Vol. 2933. Springer, 269–285.

[83] Andrew Pohorille and David Deamer. 2009. Self-assembly and function of primitive cell membranes. *Res. Microbiol.* 160, 7 (2009), 449–456.

[84] Gheorghe Păun. 2000. Computing with membranes. *J. Comput. Syst. Sci.* 61, 1 (2000), 108–143.

[85] Gheorghe Păun. 2002. *Membrane Computing: An Introduction*. Springer-Verlag, Berlin.

[86] Gheorghe Păun and Radu A. Păun. 2006. Membrane computing and economics: Numerical P systems. *Fundam. Inform.* 73, 1–2 (2006), 213–227.

[87] Gheorghe Păun, Grzegorz Rozenberg, and Arto Salomaa (Eds.). 2009. *The Oxford Handbook of Membrane Computing*. Oxford University Press.

[88] Juan Quiros, Sergey Verlan, Julian Viejo, Alejandro Millán, and Manuel J. Bellido. 2016. Fast hardware implementations of static P systems. *Comput. Inform.* 35, 3 (2016), 687–718.

[89] Raúl Reina-Molina, Daniel Díaz-Pernil, and Miguel A. Gutiérrez-Naranjo. 2011. Integer linear programming for tissue-like P systems. In *Proceedings of the 9th Brainstorming Week on Membrane Computing*, M. A. Martínez-del-Amor, Gh. Păun, I. Pérez-Hurtado, F. J. Romero-Campero, and L. Valencia-Cabrera (Eds.).

[90] Haina Rong, Kang Yi, Gexiang Zhang, Jianping Dong, Prithwineel Paul, and Zhiwei Huang. 2019. Automatic implementation of fuzzy reasoning spiking neural P systems for diagnosing faults in complex power systems. *Complexity* 2019 (2019), 2635714:1–2635714:16.

[91] Grzegorz Rozenberg and Arto Salomaa (Eds.). 1997. *Handbook of Formal Languages*. Vol. 1–3. Springer.

[92] Eduardo Sánchez-Karhunen and Luis Valencia-Cabrera. 2019. Modelling complex market interactions using PDP systems. *J. Memb. Comput.* 1, 1 (2019), 40–51.

[93] James E. Smith. 1984. Decoupled access/execute computer architectures. *ACM Trans. Comput. Syst.* 2, 4 (1984), 289–308.

[94] Yasuhiro Suzuki, Yoshi Fujiwara, Junji Takabayashi, and Hiroshi Tanaka. 2000. Artificial life applications of a class of P systems: Abstract rewriting systems on multisets. In *Proceedings of the Workshop on Membrane Computing - Multiset Processing (LNCS)*, C. S. Calude, Gh. Păun, G. Rozenberg, and A. Salomaa (Eds.). Springer, 299–346.

[95] Yasuhiro Suzuki and Hiroshi Tanaka. 2006. *Modeling p53 Signaling Pathways by Using Multiset Processing*. Springer Berlin, 203–214.

[96] El-Ghazali Talbi, Sanaz Mostaghim, Tatsuya Okabe, Hisao Ishibuchi, Günter Rudolph, and Carlos A. Coello Coello. 2008. Parallel approaches for multiobjective optimization. In *Multiobjective Optimization, Interactive and Evolutionary Approaches (LNCS)*, J. Branke, K. Deb, K. Miettinen, and R. Slowinski (Eds.), Vol. 5252. Springer, 349–372.

[97] El-Ghazali Talbi. 2018. A unified view of parallel multi-objective evolutionary algorithms. *J. Parallel Distrib. Comput.* 133 (2018), 349–358.

[98] Jorge A. Tejedor, Luis Fernández, Fernando Arroyo, and Sandra Gómez-Canaval. 2007. Algorithm of rules applications based on competitiveness of evolution rules. In *Proceedings of the 8th Workshop on Membrane Computing*, G. Ekeftherakis, P. Kefalas, and Gh. Păun (Eds.). 567–580.

[99] Jorge A. Tejedor, Luis Fernández, Fernando Arroyo, and Abraham Gutiérrez. 2007. Algorithm of active rule elimination for application of evolution rules. In *Proceedings of the 8th WSEAS International Conference on Evolutionary Computing*, Akshai Aggarwal (Ed.). 259–267.

[100] Jorge A. Tejedor, Abraham Gutiérrez, Luis Fernández, Fernando Arroyo, Ginés Bravo, and Sandra Gómez-Canaval. 2007. Optimizing evolution rules application and communication times in membrane systems implementation. In *Proceedings of the 8th International Workshop on Membrane Computing (WMC'07) (Lecture Notes in Computer Science)*, George Eleftherakis, Petros Kefalas, Gheorghe Păun, Grzegorz Rozenberg, and Arto Salomaa (Eds.), Vol. 4860. Springer, 298–319.

[101] Roman Trobec, Marián Vajteršic, and Peter Zinterhof. 2009. *Parallel Computing*. Springer.

[102] Christos Tsotskas, Timoleon Kipouros, and Anthony Mark Savill. 2014. The design and implementation of a GPU-enabled multi-objective Tabu-Search intended for real world and high-dimensional applications. *Proced. Comput. Sci.* 29 (2014), 2152–2161.

[103] Ganesh Venkatesh, Jack Sampson, Nathan Goulding, Sravanthi Kota Venkata, Michael Bedford Taylor, and Steven Swanson. 2011. QSCORES: Trading dark silicon for scalable energy efficiency with quasi-specific cores. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'11)*. 163–174.

[104] Sergey Verlan. 2010. Study of Language-Theoretic Computational Paradigms Inspired by Biology. Habilitation thesis, Université Paris Est.

[105] Sergey Verlan. 2013. Using the formal framework for P systems. In *Proceedings of the 14th International Conference on Membrane Computing (CMC'13) (Lecture Notes in Computer Science)*, Artiom Alhazov, Svetlana Cojocaru, Marian Gheorghe, Yurii Rogozhin, Grzegorz Rozenberg, and Arto Salomaa (Eds.), Vol. 8340. Springer, 56–79.

[106] Sergey Verlan and Juan Quiros. 2012. Fast hardware implementations of P systems. In *Proceedings of the 13th International Conference on Membrane Computing (CMC'12) (Lecture Notes in Computer Science)*, Erzsébet Csuhaj-Varjú, Marian Gheorghe, Grzegorz Rozenberg, Arto Salomaa, and György Vaszil (Eds.), Vol. 7762. Springer, 404–423.

[107] Tao Wang, Gexiang Zhang, and Mario J. Pérez-Jiménez. 2015. Fuzzy membrane computing: Theory and applications. *Int. J. Comput. Commun. Contr.* 10 (2015), 904–935.

[108] Tao Wang, Gexiang Zhang, Junbo Zhao, Zhenyou He, Jun Wang, and Mario J. Pérez-Jiménez. 2015. Fault diagnosis of electric power systems based on fuzzy reasoning spiking neural P systems. *IEEE Trans. Power Syst.* 30, 3 (2015), 1182–1194.

[109] Xueyuan Wang, Gexiang Zhang, Ferrante Neri, Tao Jiang, Junbo Zhao, Marian Gheorghe, Florentin Ipate, and Raluca Lefticaru. 2016. Design and implementation of membrane controllers for trajectory tracking of nonholonomic wheeled mobile robots. *Integ. Comput.-Aided Eng.* 23, 1 (2016), 15–30.

[110] Xueyuan Wang, Gexiang Zhang, Junbo Zhao, Haina Rong, Florentin Ipate, and Raluca Lefticaru. 2015. A modified membrane-inspired algorithm based on particle swarm optimization for mobile robot path planning. *Int. J. Comput. Commun. Contr.* 10, 5 (2015), 732–745.

[111] Nicholas Wilt (Ed.). 2013. *The CUDA Handbook: A Comprehensive Guide to GPU Programming*. Addison Wesley.

[112] Erica Wiseman. 2016. Next generation computing. *National Research Council of Canada/Gov. of Canada* (2016). https://cradpdf.drdc-rddc.gc.ca/PDFS/unc268/p805200_A1b.pdf.

[113] Zihan Xu, Matteo Cavaliere, Pei An, Sarma Vrudhula, and Yu Cao. 2014. The stochastic loss of spikes in spiking neural P systems: Design and implementation of reliable arithmetic circuits. *Fund. Inform.* 134, 1–2 (2014), 183–200.

[114] Jianying Yuan, Dequan Guo, Gexiang Zhang, Prithwineel Paul, Ming Zhu, and Qiang Yang. 2019. A resolution-free parallel algorithm for image edge detection within the framework of enzymatic numerical P systems. *Molecules* 24, 7 (2019).

[115] Xiangxiang Zeng, Henry Adorna, Miguel A. Martínez-del-Amor, Linqiang Pan, and Mario J. Pérez-Jiménez. 2010. Matrix representation of spiking neural P systems. In *Proceedings of the 11th International Conference on Membrane Computing (CMC'10) (LNCS)*, Marian Gheorghe, Thomas Hinze, Gheorghe Păun, Grzegorz Rozenberg, and Arto Salomaa (Eds.). 377–391.

[116] Gexiang Zhang, Jixiang Cheng, Marian Gheorghe, and Qi Meng. 2013. A hybrid approach based on differential evolution and tissue membrane systems for solving constrained manufacturing parameter optimization problems. *Appl. Soft Comput.* 13, 3 (2013), 1528–1542.

[117] Gexiang Zhang, Marian Gheorghe, Linqiang Pan, and Mario J. Pérez-Jiménez. 2014. Evolutionary membrane computing: A comprehensive survey and new results. *Inform. Sci.* 279 (2014), 528–551.

[118] Gexiang Zhang, Mario J. Pérez-Jiménez, and Marian Gheorghe. 2017. *Real-life Applications with Membrane Computing* (1st ed.). Springer Publishing Company, Incorporated.

[119] Gexiang Zhang, Haina Rong, Ferrante Neri, and Mario J. Pérez-Jiménez. 2014. An optimization spiking neural P system for approximately solving combinatorial optimization problems. *Int. J. Neural Syst.* 24, 5 (2014), 1440006.

[120] Weihang Zhu, Ashraf Yaseen, and Yaohang Li. 2011. DEMCMC-GPU: An efficient multi-objective optimization method with GPU acceleration on the Fermi architecture. *New Gen. Comput.* 29, 2 (2011), 163–184.