An Overview of Hardware Implementations of P Systems

Zeyi Shang[1], Sergey Verlan[2], Gexiang Zhang[1]*,
Miguel A. Martínez-del-Amor[3], and Luis Valencia-Cabrera[3]

[1] School of Electrical Engeneering,
Southwest Jiaotong University,
Chengdu, Sichuan, China
{sibanordol,zhgxdylan}@126.com
[2] Université Paris Est, LACL (EA 4219), UPEC, F-94010, Créteil, France
verlan@u-pec.fr
[3] Research Group on Natural Computing
University of Seville, Sevilla, Spain
{mdelamor,lvalencia}@us.es

Abstract. Implementing the P systems on parallel hardware is a research highlight in bio-inspired computing since the membrane computing is a large-scale parallel computing paradigm which have a potential to tremendously speed up the computation. Field-programmable gate arrays (FPGAs) and CUDA-enabled GPUs are the primary hardware which is employed to implement P systems. FPGA-based hardware implementations use different strategies considering regions or evolution rules as processing units. This implies the existence of several parallel architectures for FPGAs specially designed to implement P systems. In contrast, the CUDA-enabled GPUs are a pre-defined parallel platform and numerous types of P systems are directly implemented on it.
The object distribution problem (choosing which rules will be applied) is the core problem of all hardware implementations. This problem is particularly difficult, because in the general case the model of P systems is non-deterministic and maximally parallel, hence the corresponding problem is NP-hard. Several heuristics were proposed in order to accelerate the process of the computation of the corresponding ruleset.
In this article we overview different approaches and designs for hardware implementations of P systems as well as corresponding solutions to the object assignment problem.

Keywords: Membrane Computing, P Systems, hardware implementation, FPGA, CUDA

A Introduction

With the putative arising of *protocells* 3.8–4 billion years ago in hydrothermal vent precipitates, at which a time-line not long after the estimated forming of the the Earth and oceans 4.54 and 4.41 billion years ago [15], the unicellular organisms which are deemed as the most primitive

* Corresponding author

lifeforms have existed and evolved until now [51]. From monoplast to multicellularity, the undisputed form of lives remain unchanged since then. That is, the biological cell structures defined by the membranes have evolved so being optimized for billions of years. As a consequence, a bio-cell is powerful parallel processing unit which can perform sophisticated biologic behaviors. Enlightened by the insights of biological membranes and the bio-chemistry reactions inside, and on the foundation of theoretical computer science, membrane computing was initiated by Gheorghe Păun in 1998. Due to its background inspired from, membrane computing has worked as a modeling framework for biological and ecological objectives such as artificial life [60], photosynthesis [49], protein signaling pathway [61], etc. On the other hand, the inherent large scale parallelism of membrane computing has the profound potential for the progress of extreme data processing. Implementing membrane computing models called P systems on contemporary silicon integrated circuits to exploit the desirable parallel computational capability of P systems to explore a new orientation for high performance computing (HPC) is what we concern about.

Pursuing the parallel computing is an imminent requirement for the time being. The density of electronic ingredients in the integrated circuit board and the corresponding computational capability has been subject to the loose Moore's law for decades. What more make sense is the fact that the computational capability is determined by the density of electronic components. However, Moore's law will not hold for long since the physical limit, which the augmenting of density is unsustainable and the subsequently awkward thermal dissipation problem, is around the corner. Even with the further increase of the density, the computational capability growth is not linearly proportional to the density [69]. Hardware engineers are missing the good old days when what they should do is just double the density to double the computational capability by the prestigious law. 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 multiple processing nodes computing. This possible conclusion stems from the evolution of computer processor scheme, from single-core single CPU to multiple-core single CPU and multiple-core multiple-CPU frame. In this sense, the inherent parallelism infers that P systems belong to a class of multiple processing nodes computing devices. What constituents work as processing nodes will give rise to different implementing strategies. The way how living cells allocate, organize and coordinate processing nodes has evolved for billions of years. Investigating this magnificent course will illuminate us to handle the multiple cores computing, which has cut a striking figure in the contemporary parallel computing realm.

The large scale distributed parallel process occurring in vesicle compartments and the vesicle division functionality enlightened from mitosis of living cells which increase the artificial cells exponentially are two of the most outstanding advantages of membrane computing that would underlie the foundation for the construction of highly parallel computation platform whose performance, flexibility and scalability outperforms traditional sequential counterparts substantially. As a parallel computing paradigm inspired by the structural and functional features of biological membranes, only the parallel computing platforms are suitable for the implementation of P systems with respect to the fact that 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 a enormous computing potential like creation and dissolution of inner membranes, the self-replication or *autopoiesis* [16] of the whole cell-like entirety that works as a computing unit, the *symport* and *antiport* of objects, etc. Implementing this new computing paradigm which exhibits a promising prospect on parallel platforms so that put this theoretical excellent performance into practical application is what pursued. It is remarked that programming the membrane computing algorithms with high level general purpose language and executing them on the computer is just simulating, not real implementing [52] of P systems.

In fact, software-based and hardware-based parallel computing platforms have developed for implementing P systems. The software-based parallel computing platform is constructed on a cluster of computers [14]. This platform achieves good performance and flexibility for that the CPUs are executing the operations and the changing of objectives is easily carried out by programs. Nonetheless, with the size of objective P system increasing, the consumption of time and CPU resources caused by the communications of different computers rising dramatically. Moreover, the underlying hardware (a cluster of computers) of this platform cannot be miniaturized so that the membrane computing algorithms cannot be utilized in embedded chips and compact controllers which can be employed in robots, automobiles, machine tools, etc. This disadvantage will limits the range of applications of membrane computing. While the hardware-based platforms are fabricated on integrated circuits. The corresponding algorithms are mapped to digital circuits and the performance is much higher compared with the software-based platforms, although this high performance may comes at the cost of flexibility and extensibility. But the reprogrammable hardware turns the corner, lifting barriers to devise portable and embedded membrane processors which can used as CPUs, controllers or something like that. This is necessity and importance of hardware implementation of P systems.

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: (1) Field-programmable gate arrays (FPGAs) and (2) CUDA-enabled graphical processing units (GPUs). In the first case a completely new parallel circuit is specially designed to simulate 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 lower in this case, but the development effort is much lower, so finally it becomes an interesting compromise between traditional computers implementations and highly parallel using specialized circuits.

The core problem for the implementations is the *object distribution problem* (ODP) that based on the current configuration yields a multiset of applicable rules that will be applied in order to pass to the next configuration. This problem is a particular variant of a more general problem that computes the applicable set of multisets of rules for a configuration and it is known to be NP-complete [13]. We recall that in the general case the model is non-deterministic, so an equitable choice among different possibilities should be provided. Known algorithms and heuristics do not parallelize well, so special heuristics were developed in order to quickly compute the desired multiset of rules.

Since the ice-breaking work of [32], multiple concerned research groups have engaged in the hardware implementation of P systems. In this article, the comprehensive and systematic analysis of hardware implementation course is presented, especially formulating the root cause of parallelism and non-determinism, and classifying and summarizing the proposed approaches under the hypothesis that P systems are multiple processing nodes models. This overview of the domain can play the role of a guideline for the future scientific explorations towards the hardware implementation of membrane computing. This paper is organized as follows: Section B gives a short description of the capabilities and the architecture of used hardware, Section C gives the definition of the most general model of P systems using the formal framework [23]. Section D presents an overview of different simulation approaches, including the DND algorithm (which is the base for all CUDA simulators). Next, Section E expresses ODP in terms of integer linear programming. Section F to Section 8 give more details on existing simulation approaches. Finally, conclusions and future research directions are discussed in Section 9.

B The Selection of Hardware

The extensive application of multiple cores computing has already pushed forward the world into the parallel computing era. As another parallel computing paradigm, the membrane computing provides a new possible scheme for parallel computation. Among the characters P systems possessing, the parallelism is the most dramatic one under the foundation of proved computational universality. The software simulation of membrane computing algorithm on the general computer which is not a highly parallel platform is far from the ideal facility to exploit the potential large scale parallel computation capacity. Implementing the parallelism and other features of P system is a set of formidable undertakings requiring delicate thoughts and excellent hardware. In the light of the thought that implements the parallel algorithm on a parallel computing device, filter out suitable apparatuses from the voluminous categories of hardware is one of the essential assignment. In short terms, we need parallel hardware architectures to execute P systems. From another perspective, if we do not select particular hardware at first, the pre-designed implementing strategies are groundless, for the correspondences between ideas and hardware components are unknown. FPGAs, CUDA-enabled GPUs and microprocessors are chose to implementing variants of P systems.

B.1 FPGA hardware

For the implementation of membrane computing models in hardware, the particular difficulties do not only lie in the realization of maximal parallelism and non-determinism (which are mainly two attributes derived from the inherent nature of biological cells that inspired P systems), but also come from the fact that there exists a big number of variations of the basic model of P systems having quite distinct characteristics [54]. This poses a great challenge for the conception of a general computational architecture to implement these various models. With the advent of reconfigurable hardware the elaboration of different types of computational models on the same reprogrammable devices is no longer an impossible assignment. The field-programmable gate arrays (abbreviated as FPGAz hereinafter) is a reconfigurable hardware allowing to prototype digital circuits. This property makes it a unique alternative for the implementation of membrane computing. The framework of membrane computing constructed on FPGA basis can be a truly parallel computing platform different from the sequential one, which just simulates the parallelism, without really implementing it. The modification of circuits for FPGAs is performed by altering the codes defining the circuits using a hardware description language. Figure 1 shows the internal structure of an FPGA.

The CLBs work as containers for slice-organized logic cells for the rapid increasing densities of logical cells. Basically, the logic cells consist of look-up tables (LUTs) and flip-flops (FFs). The LUT is used for realizing various combinatorial circuits such as basic gates, decoders, encoders and multiplexers. The FF is the sequential logic component acting as a storage element for the circuit. In fact, what can be reconfigured is not the CLBs but the interconnects. As result, CLBs just like stationary islands stand in the undulant interconnect sea. FPGA interconnect technology consists of switch boxes that route signals between various logic blocks of it. Present-day FPGAs are based on one of the following interconnect technologies: static RAM, flash memory or anti-fuse. The first one dominates the current FPGAs. Figure 2 illustrates the schematic diagram of a logic cell and a slice. With the re-programmability, FPGAs comply with the model-oriented hardware implementation quite well. The parallelism, non-determinism and other features can be realized on FPGAs by the particularly designed circuits.

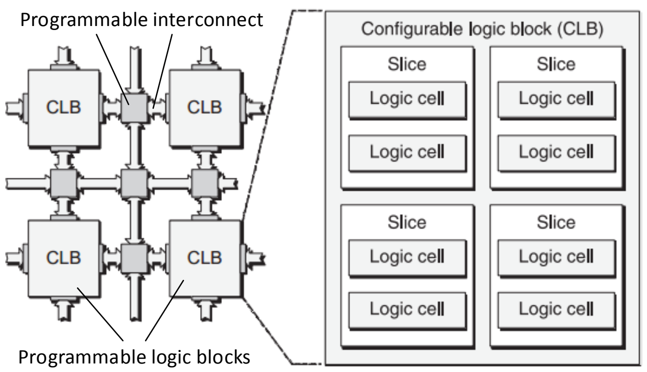
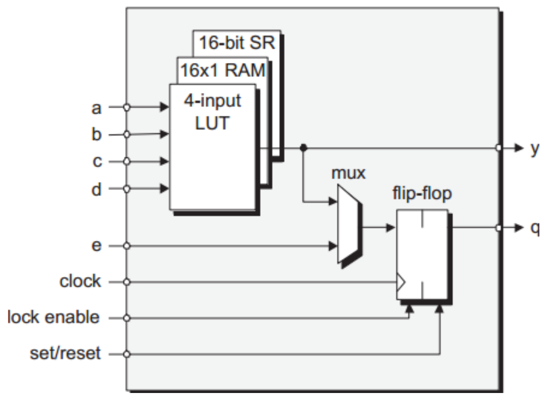
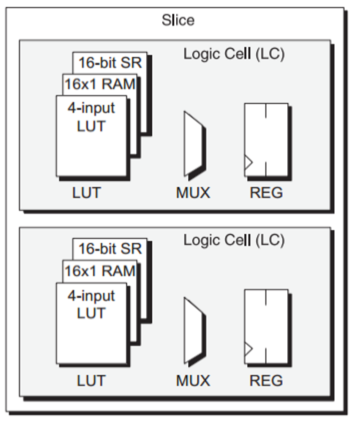


Fig. 1: A FPGA's internal fabric. The structural feature of a FPGA is that the configurable logic blocks (CLB) are inlaid in the matrix of interconnects. In this type, the each CLB contains four slices, each of which includes 2 logic cells [42].



(a) A simplified view of a logic cell



(b) A slice containing two logic cells

Fig. 2: The schematic diagram of a logic cell and a slice from [42].

B.2 CUDA-enabled GPU hardware

The multiple computing cores solution is introduced from the mainframe computer field to improve the clock speed. Nowadays, it is hard to find a PC equipped with only one CPU. On the other side, 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) [30]. 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 concern 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 which can handle other computing assignments except for graphics processing. The predicament has changed for the arise of *compute unified device architecture*, known as *CUDA*, from the 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 referred is not the parallel computing framework but a GPU supporting CUDA. A CUDA-enabled graphics processing unit is an universal parallel computing device which 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. CUDA programming model is based on *heterogeneous computing* [30], 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 (see Figure 3). 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. The execution of threads inside a block can be synchronized by *barrier* operations, and threads of different blocks can be synchronized only by finishing the execution of the kernel.

The memory hierarchy is explicitly managed (see Figure 3). Although current GPUs contain cache memories, in order to accelerate memory accesses, best performance is achieved when doing it manually. A GPU basically contains a *global memory*, which is the largest but the slowest memory in the system, and *shared memory*, which is the smallest but fastest memory [30]. 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 a block. Threads also have fast access to its own registers for single variables, and local memory (which is normally outsourced to global memory). Accesses to memory has to be carefully programmed, so that contiguous portion of data are read by consecutive threads (*coalesced access*), since this increases the memory bandwidth utilization.

Nowadays,the architecture of GPUs is upgraded to *Streaming Multiprocessors (SMs)* which are composed of an array of *Streaming Processors (SPs)*, working as computing cores. A thread set consists of 32 threads named *warp* is the basic unit which a SM fulfills its executions. A SM can manage multiple warps which 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 [38]. The SM framework and its warp flow is shown in figure 4.

B.3 Other hardware

Custom application specific integrated circuits (ASIC) can also be employed to implement P systems. Some researches investigated simulating certain P systems on micro-processors [27].

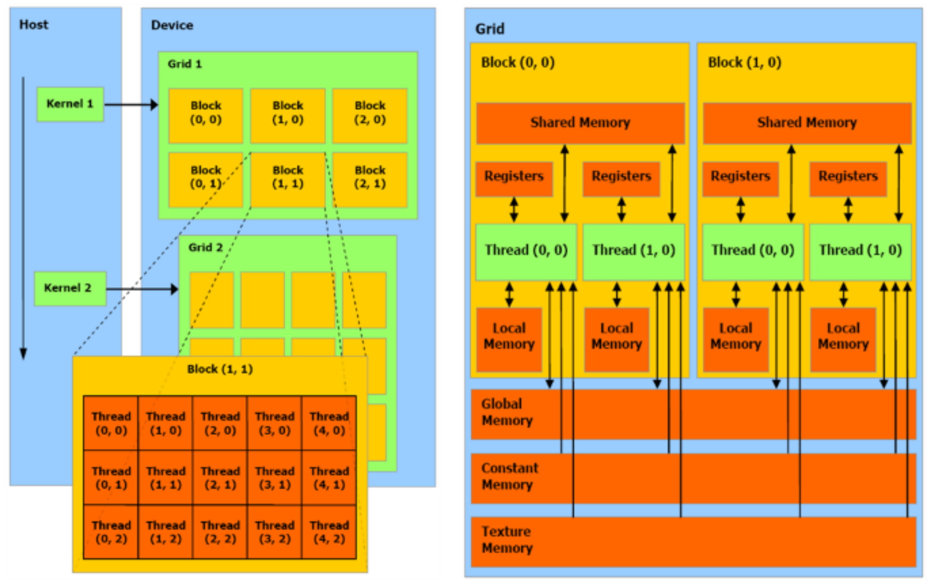


Fig. 3: (left) CUDA execution model, (right) CUDA memory model. From [1].

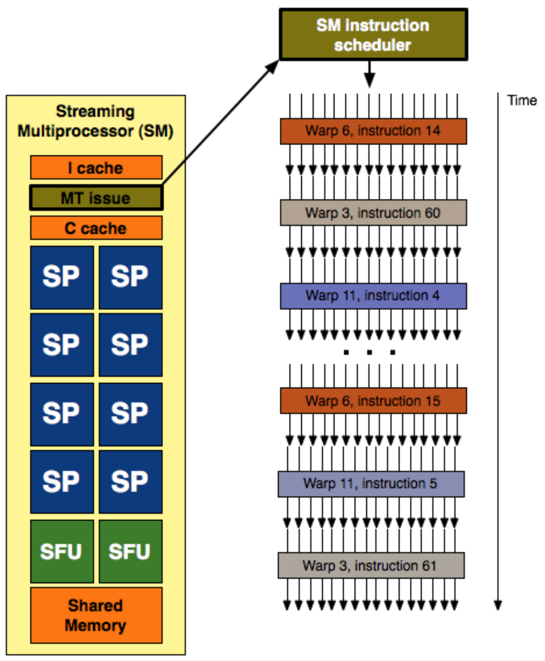


Fig. 4: A SM is composed of an array of SPs, shared memory and a couple of caches. A MT issue module contains a SM instruction scheduler which conducts instruction flows.

Although the performances of micro-processors are low, they are economical alternatives suitable for the developing of prototypes and verifying the design methods. However, as stated, membrane computing model is a sort of machine-oriented model for the various types. The flexibility of the hardware platform constructed on ASIC is quite insufficient to adapt to the different variants of P system. However, such attempts are still interesting as they allow to simulate features of P system on some tailored circuits with different interesting properties, like low power consumption.

C The model of P systems

We assume that the reader is familiar with basic notions of formal language theory and membrane computing; for further details consult [53] and [56].

For a finite multiset of symbols M over an alphabet V, $supp(M)$ denotes the set of symbols in M (the support of M) and $|M|$ denotes its size, i.e., the total number of its symbols. By $|M|_x$, the number of occurrences of symbol x in M is denoted. By V° we denote the set of all finite multisets over V.

Throughout the paper, every finite multiset M is given as a string w, where M and w have the same number of occurrences of symbol a, for each $a \in V$.

C.1 Network of Cells

To give a more precise description of the semantics of a P system, the following notions (functions) were defined:

- $Applicable(\Pi, \mathcal{C}, \delta)$ – the set of multisets of rules of Π applicable to the configuration $\mathcal{C}$, according to some derivation mode δ.
- $Apply(\Pi, \mathcal{C}, R)$ – the configuration obtained by the (usually parallel) application of the multiset of rules R to the configuration $\mathcal{C}$.
- $Halt(\Pi, \mathcal{C}, \delta)$ – a predicate that yields true if $\mathcal{C}$ is a halting configuration of the system Π using the derivation mode δ.
- $Result(\Pi, \mathcal{C})$ – a function giving the result of the computation of the P system Π when the halting configuration $\mathcal{C}$ has been reached. Usually, this is an integer function. However, generalizations as for example, Boolean or vector functions can also be considered.

We note that δ, above, differs from the dissolution symbol used in some P system models.

The precise interpretation of the four notions (functions) above depends on the chosen model of P systems. The goal of works [22, 23, 65] was to provide a concrete family of P systems based on the structure of *network of cells* together with a series of definitions of the functions above. The obtained model as well as the accompanying tools and methods together are called the *formal framework of P systems*. It has the property that most of the existing models of P systems could be obtained by a strong bi-simulation of a restricted version (eventually, using a simple encoding) of this formal framework with respect to different parameters, see [66] for some examples.

We would also like to note that based on the formal framework it is possible to reduce the structure of any P system to one membrane. The corresponding process is called flattening and it is described in more details in [21, 23].

Below, we provide a summarized version of the definition of a *network of cells*, the class containing all networks of cells forming the structure of the formal framework. The definitions are based on those given in [23]. This version considers only static P systems where the membrane structure does not change under the computation (this also includes systems with the dissolution of membranes). We note that in [22], an extension of the formal framework to P systems with dynamically evolving structure is proposed. We remark that in the case of static structures both variants coincide, although the notation is slightly different.

Definition 1 **([23]).** *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. *$w = (w_1, \dots, w_n)$ where $w_i \in V^\circ$, for all $1 \leq i \leq n$, is the* finite multiset initially associated to cell *i;*
4. *$Inf = (Inf_1, \dots, Inf_n)$ 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 $X = (x_1, \dots, x_n)$, $Y = (y_1, \dots, y_n)$, $x_i, y_i \in V^\circ$, $1 \leq i \leq n$, are vectors of multisets over V and $P = (p_1, \dots, p_n)$, $Q = (q_1, \dots, q_n)$, p_i, q_i, $1 \leq i \leq n$ are finite sets of multisets over V. We will also use the notation
$$(1, x_1) \dots (n, x_n) \to (1, y_1) \dots (n, y_n) ; [(1, p_1) \dots (1, p_n)]; [(1, q_1) \dots (n, q_n)]$$
for a rule $(X \to Y; P, Q)$; moreover, if some p_i or q_i is an empty set or some x_i or y_i is equal to the empty multiset, $1 \leq i \leq n$, then we may omit it from the specification of the rule.

The semantics of the above rule is as follows: objects x_i from cells i are rewritten into objects y_j in cells j, $1 \leq i, j \leq n$, if every cell k, $1 \leq k \leq n$, contains all multisets from p_k and does not contain any multiset from q_k. In other words, the first part of the rule specifies the rewriting of symbols, the second part of the rule specifies permitting conditions and the third part of the rule specifies the forbidding conditions.

For a rule r of the form above, the set

$$\{i \mid x_i \neq \lambda \text{ or } y_i \neq \lambda \text{ or } p_i \neq \emptyset \text{ or } q_i \neq \emptyset\}$$

induces a (hypergraph) relation between the interacting cells. However, this relation does not need to give rise to a *structure* relation like a tree as in P systems or a graph as in tissue P systems.

A *configuration* C of Π is an n-tuple of multisets over V $(u_1, \dots, u_n)$ satisfying $u_i \cap Inf_i = \emptyset$, $1 \leq i \leq n$.

P systems work with transitions between configurations; a finite sequence of such transitions of a P system Π starting with the initial configuration and ending in some final configuration is called a computation. The final configuration is usually given by halting.

The transition of a P system Π according to the derivation mode δ (usually, the maximally parallel derivation mode) is defined as follows: the system changes from a configuration $\mathcal{C}$ to $\mathcal{C}'$ (written as $\mathcal{C} \Rightarrow \mathcal{C}'$) iff

$$\mathcal{C}' = Apply(\Pi, \mathcal{C}, R), \text{ for some } R \in Applicable(\Pi, \mathcal{C}, \delta)$$

The result of the computation of a P system is usually interpreted as the union of the results of all possible computations.

D Simulation of P systems using hardware

When one speaks about the hardware implementation of membrane computing, how to characterize the membrane structures in hardware circuits is the most concerned issue, followed by the representation of contents included in membranes: multisets of objects and evolution rules. The implementation of parallelism, non-determinism and other attributes of P systems is realized on the basis of the hardware characterization of membrane structures and their substances.

D.1 The Characterization of Membrane structures

Indeed, there are no compartments in silicon integrated circuits which can be used as membranes, let alone the specific functions on account of the arising of membranes, for instance, the traversing membrane behaviors of objects. Nevertheless, according to [52], membrane is just an idealized concept without internal structures analogous to biological ones. The main functionality of membranes are working as boundaries and the delimited space caused by the presence of membranes makes P systems as distributed computing models. The spatial placement and size of membrane is not important, except the inter-relationship among them. In view of this fact, the representation of three-dimension arrangement of vesicles for membranes can be transformed to two-dimension Venn diagram expression. This flattening process is favorable for understanding the essential function of membranes in the sense of hardware implementation.The breaking point of the conundrum is the one-to-one correspondence between membrane and its enclosed region, which contains multiset of objects and evolution rules. Then an injective correspondence between membrane and its contents can be established, which is used in the hardware implementation to represent the whole cellular construction. Speak straightforwardly, the membrane structure is transformed to its substances distributed in the hardware which implements the P system. From the theoretical point of view, any membrane structure can be reduced (flattened) to just a single membrane, see [21, 23] for more details.

D.2 The Presentation of Multisets of Objects

After transforming the membrane structure into a matter that stands a chance to implement it on a hardware, there are two problems to deal with for the representation of the multisets of objects in a region: the distinct symbols from the alphabet representing object types and its corresponding multiplicities. The strategies adopted for the two problems are storing the multiplicities in different registers of hardware device. Specifically, an array of registers is employed to store each of the multiplicity value which is a positive integer in a position of the array and the different locations of registers denote distinct symbols naturally [50]. In view of possible objects traversal in future evolutions between regions, the number of registers of each array should equal to the number of object types in the P system. In consequence, the multiplicities of some objects are zeros in initial configuration and the corresponding array is sparse.

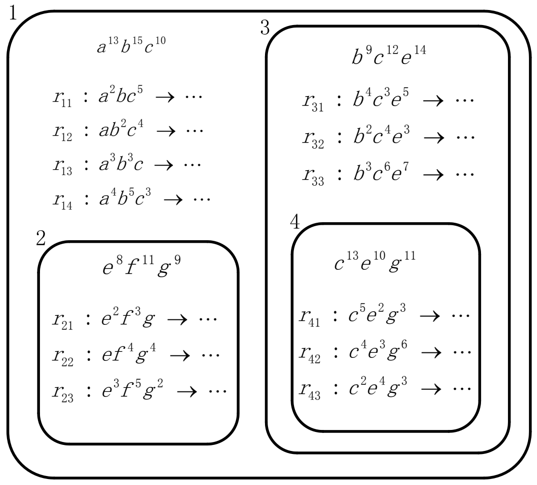
D.3 The Presentation of Evolution Rules

The multisets of objects serve as the reactants for evolution rules inside membranes. The consequence of applying a rule is reflected by the alteration of multiplicities of objects in associated regions. From an intuitive point of view, a rule can be represented by storing the two sides in registers separately [50]. Once the rule is applied according to its instance number, arithmetical operations of plus and minus commence to increase and decrease multiplicities of relevant objects in relevant registers.

Rules remain unchanging along with the whole evolving process of configurations until to the halt condition. Take into consideration that rules are incorporated in regions, to some extent, the function of rules can be regarded internalized to the regions, i.e., the explicit expression of rules is the interpretation of the effect of regions. In light of the conceived idea towards the relation of rules and regions, mapping a processing unit which conducts the execution of a set of rules to each region [44] is a viable thought. From analogous perspective, mapping a processing unit to each rule [47] so that the modification of multiplicities can be committed by rules straightforwardly is also viable. Under this circumstance, the region is consolidated by rules enclosed. The two fundamental principles are exactly utilized to develop the hardware pattern to execute P systems. In particular, the two principles motivate the region-based simulation and rule-based simulation approach, which will be formulated below.

D.4 The Analysis of Parallelism and Non-determinism of P Systems

There are two parallelism meaning in P systems: the region-level parallelism which stems from applying multiple rules concurrently and system-level parallelism which derives from the parallel executing of region-level parallelism in all the regions [46]. The organization and execution of applicable rules subject to certain parallel derivation mode, which brings about the region-level parallelism, i.e., what applied is not a single rule but a multiset of rules in a given region. This fact lead to the parallelism, which is implemented in hardware by synchronization operations. Usually, several multisets of rules meet the applicability and derivation mode requirements. Each region should choose one multiset of rules to evolve configuration. Consequently, the non-determinism emerges. In general, it is a multi-solution problem to combine rules in terms of parallel derivation mode in every region to evolve the the configuration. To dispose of this problem, the living cells adopt non-determinism to select one multiset of rules from the alternatives. Furthermore, choosing and applying the multiset of rules in every region non-deterministically give rise to the non-deterministic evolving of configurations. The common strategy adopted for implementing non-determinism is assigning each multiset of rules equi-probability. To articulate the parallelism and non-determinism in P systems, a static P system whose membrane structure does not change during the transition of configurations is introduced, shown in figure 4.


1
$a^{13}b^{15}c^{10}$
r_{11} : $a^2bc^5 \rightarrow \cdots$
r_{12} : $ab^2c^4 \rightarrow \cdots$
r_{13} : $a^3b^3c \rightarrow \cdots$
r_{14} : $a^4b^5c^3 \rightarrow \cdots$
2
$e^8f^{11}g^9$
r_{21} : $e^2f^3g \rightarrow \cdots$
r_{22} : $ef^4g^4 \rightarrow \cdots$
r_{23} : $e^3f^5g^2 \rightarrow \cdots$
3
$b^9c^{12}e^{14}$
r_{31} : $b^4c^3e^5 \rightarrow \cdots$
r_{32} : $b^2c^4e^3 \rightarrow \cdots$
r_{33} : $b^3c^6e^7 \rightarrow \cdots$
4
$c^{13}e^{10}g^{11}$
r_{41} : $c^5e^2g^3 \rightarrow \cdots$
r_{42} : $c^4e^3g^6 \rightarrow \cdots$
r_{43} : $c^2e^4g^3 \rightarrow \cdots$
Fig. 5: A Configuration of a static P System. The multiset of objects and evolution rules contained in each region are illustrated. The right-hand side of each rule is not given since it is not required for identifying its applicability.

One of the most used derivation modes is *maximal parallelism* mode, abbreviated as *max*. Informally, consuming the ceiling number of objects is the goal of maximal application of rules, which means that the combinations of applicable rules consume objects as many as possible, until exhausting some kinds of objects, or the remaining objects are not enough for one more applying of any rules. The sets of multisets of rules calculated in terms with *max* mode in each region of the P system are shown in figure 5. object The application of rules is a course to consume objects. This course can be abstracted as a object distribution problem which allocates objects to applicable rules. Develop approaches to converge to all the feasible multisets of rules associated with current configuration accurately and efficiently and utilize them non-deterministically to evolve the system are the two core tasks. Specifically, in the context of parallel and non-deterministic computing, it needs to be clear that the goal of the approach for the object distribution problem is to acquire a serial of sets composed of multisets of rules constructed coinciding with the parallel derivation mode. Each such set corresponding to a region of the P system and a solution is the group of all the sets obtained in every region. We will illustrate the parallelism and non-determinism systematically as follows.

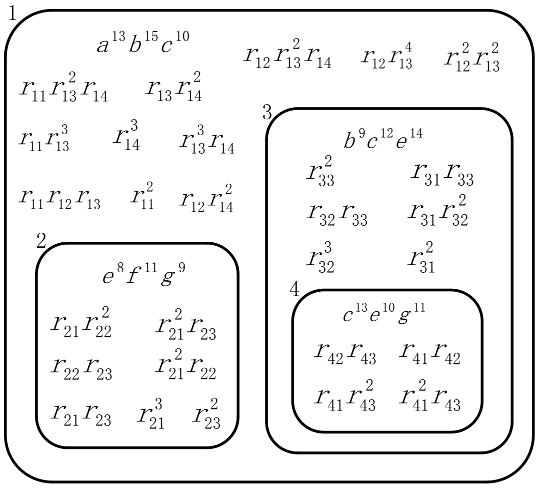


Fig. 6: The set of multisets of rules in each region. All the multisets of rules in a region corresponding to a *regional solution*, a single multiset of rules in a region denotes a *regional feasible multiset*. All regional solutions comprise the *complete solution*, and the set of selected regional feasible solutions from each region comprise the *dynamic solution*.

Consider the following example depicted on Figure 6. In region 1 the multiset of rules is of form $r_{11}^{\alpha}r_{12}^{\beta}r_{13}^{\gamma}r_{14}^{\delta}$, where $\alpha \in [0,2], \beta \in [0,2], \gamma \in [0,4], \delta \in [0,3]$. Of course, all the indexes are integers in their range. The upper boundary is calculated without taking in consideration other rules, according to *max* mode. The combination space of the four rules is composed of 180 $(C_3^1C_3^1C_5^1C_4^1)$ compositions (zero rule is regarded as a composition). However, the set of multisets of rules depicted in figure 6 shows that only 11 combinations [45] are valid for the object distribution problem. There seem to be dependencies between a selected rule and its predecessors that influence the subsequent rules' composition. It can be assumed that the dependency and the ceiling number of instance stem from the limited quantity and variety of objects.

We would like to remark that it is not necessary to compute the set of multisets of (maximally parallel) applicable rules. It is sufficient to obtain one of them, according to some hypotheses, e.g. of equiprobable distribution. Next, the corresponding multiset is applied and the process is repeated for the new configuration. Hence, it is important to clearly define the notion of the solution to ODP. The set of multisets of applicable rules in each region is called a *regional solution*, while a single multiset of rules from the aforementioned set is referred as a *regional feasible multiset*. The collection of regional solution in all regions is defined as *complete solution*. For the evolution purposes, from each region a random feasible solution is selected. The entire of the selected regional feasible solutions is called the *dynamic solution*. If the considered P system is represented from the multiset rewriting system with only one skin membrane, the *complete solution* and *dynamic solution* degenerate to the *regional solution* and a *regional feasible multiset* respectively.

It is worthwhile to note that the *complete solution* consists in the outset of a particular configuration. Once a dynamic solution executes, the configuration evolves and a new *complete solution* is computed. Do we have to compute the complete solution? The answer is negative considering the Pyrrhic is not worth the loss for calculating the *complete solution* with respect to the time and hardware resource consuming. Whereas, it needs to be emphasized here that although a dynamic solution is computed to evolve the system, it dose not imply that the complete solution should be refused to take into consideration. This statement is on account of the fact that the non-determinism will decline if the feasible solutions always belong to the same proper subset of complete solution. From a intuitive perspective, this reduction of non-determinism is non-positive. We need calculate the dynamic solution with minimal effort while do not impair the degree of non-determinism.

An efficient way to compute and represent the dynamic solution from the *complete solution* (denoted as $Applicable(\Pi, C, \delta)$ in [54]) is required. The difficulty of the problem is not only to compute the *complete solution*, but also to ensure that it is non-deterministically chosen among all feasible solutions. The common practice is to assign the same probability to each feasible solution. In most of the cases investigated in the literature, this aspect is neglected and it is not possible to affirm that this property holds. The Direct Non-deterministic Distribution (DND) algorithm introduced in [45] tries to take this aspect into account, although without giving a formal proof. In [54, 67] a truly non-deterministic evolution is implemented, however the corresponding implementation is limited to some particular derivation modes.

Indirect approaches As in [45] we will classify different algorithms for the multiset of rules to be applied in *direct* and *indirect* ones. In the direct approach, the corresponding multiset is directly constructed by the algorithm. 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, an 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. 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 the algorithm is exploring different combinations until it reaches a valid one, it will be classified as indirect.

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 [20]. Before the first correct solution is obtained, some invalid solutions should be rejected. This approach is called *indirect straightforward approach* [45]. 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 [45, 47] called *indirect incremental approach* investigates a strategy generating possible solutions in rounds. This course can be detailed as follows.

All the applicable rules verified in accordance with the multiset of objects in a region are laid in a pool [35]. The maximal instance of each rule is calculated as well. In the first round, the instance number of a rule selected randomly is assigned a random value which contained in its range. After that the multiplicity of objects in the configuration is updated accordingly to the number of instances of the rule. Then another rule from the rest of rules is randomly picked out randomly and assigned a random number in its range, consuming the amount of objects accordingly and updating multiplicity value. This process is repeated until the last rule is reached. The number assigned to each rule's instance value is preserved until this rule is no longer considered. In the second round, the number of instances of every randomly selected rule is increased by a random amount, but keeping the number of rule instances below the maximally possible amount. When the number of rule instances cannot be increased anymore, it is removed from the pool. The whole process is repeated until the pool is empty. At each round the current multiset of rules is checked if it represents a solution. This allows to stop the process in advance. Clearly, the described algorithm allows to compute a random solution, however with an unclear distribution. Also, it is not possible to use this algorithm to compute all possible solutions as the number of instances of a rule can only increase.

Earlier variants of the indirect approach used to increment the multiplicity of one randomly chosen rule at a time [17], but this proved to be time consuming. Another attempts based on a similar idea but with different rule elimination strategies were done in [18, 26, 29, 58, 62, 63].

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.

In the *direct straightforward 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 [45] is argued that such approach is infeasible for an arbitrary configuration and rule types, it can still be applied in a big number of cases. As shown in [54, 67], if at each step the number of solutions can be expressed as a 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 a way similar to the representation of a number in the combinatorial number system). In [67] 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 [45]. A similar algorithm can also be found in [57]. This algorithm works in 2 phases. At the first phase all rules (initially randomly shuffled) except one are selected to be applied a random number of times between 0 and its maximal applicability value. In the second phase, all the rules beginning by the one excluded of the previous phase are applied to its maximal applicability benchmark in the converse order.

The DND algorithm was never implemented in FPGA hardware. However, a variant of it became popular in the simulation of Population Dynamics P (PDP) systems, named DND-P [41]. In CUDA-based development, the evolution of the DND-P algorithm, DCBA [40], was employed for the engine of the PDP system simulator [39].

One of the difficulties of the above approaches is the handling of the 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 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 [54, 67]. We recall that it is a direct approach – for any configuration the cardinality of $Applicable(\Pi, C, \delta)$ can be computed in constant time and then a random number between 1 and this cardinality is uniformly selected and the corresponding solution is decoded from this number. We also remark that this algorithm was implemented in FPGA hardware and yielded a very good performance.

E Reduction to Diophantine Equations and Integer Linear Programming

In this section we show the reduction of the problem of the computation of an element from $Applicable(\Pi, C, max)$ to the problem of solving a system of Diophantine equations or to integer linear programming (ILP).

We start with the remark that several attempts were done to express the maximally parallel choice of rules using Diophantine equations or ILP. In [2, 55] a simple variant of ILP was used (corresponding to the equations (1a) below and variable sum as maximization function). However, in this case only maximally parallel rulesets having a maximal number of rules are obtained. In terms of the formal framework [23] this corresponds to $max_{rules}max$ derivation mode. Another attempt using ILP was done in [12, 13] where the objective function is a weighted sum (by the number of objects in the left-hand-side). Such ILP 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 [45] the set of maximally parallel multisets of rules is expressed as solutions of a system of Diophantine equations (roughly equations (1a) from below) with an additional constraint to be satisfied on a solution, expressed as another system of Diophantine equations. Finally, in [5] 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 ILP.

For simplicity, we consider that Π has only one membrane (and no environment). If this is not the case, we can apply the flattening procedure reducing it to one membrane [21, 23]. So, $\Pi = (O, w_1, R)$.

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

We will construct a system of inequalities whose integer solutions will be maximally parallel multisets of rules for the configuration C. For a solution, the set of variables x_i, $1 \leq i \leq m$ indicate the cardinality of corresponding rules in some maximally parallel multiset $\mathcal{M} \in Applicable(\Pi, C, max)$.

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

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

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

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

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

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 lhs) of rule cardinalities is smaller or equal than the number of objects a in C.

Inequalities (1b) state the maximality property of the rule set defined by $x_1, \ldots, x_m$. It verifies that for each rule there exist 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 (1c) ensure that only one constraint from (1b) will be considered (the other ones will be satisfied because of M).

Hence, any solution $x_1, \ldots, x_m$ satisfying the system of inequalities (1) corresponds to a maximally parallel rule set $\mathcal{M} = r_1^{x_1} \ldots r_m^{x_m}$ applicable to configuration C. From the construction given above, it immediately follows that system (1) is Diophantine.

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

Derived from (1a):

$$x_1 + x_2 \leq 2$$
$$x_1 + x_3 \leq 3$$
$$x_1 \leq 2$$

Derived from (1b) for r_1:

$$x_1 + x_2 + 1 + M z_a^1 \geq 3$$
$$x_1 + x_3 + 1 + M z_b^1 \geq 4$$
$$x_1 + 1 + M z_c^1 \geq 3$$

Derived from (1c) for r_1:

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

Derived from (1b) for r_3:

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

Derived from (1b) for r_3:

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

It is not difficult to see that this system has 3 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 (1) might be tedious to obtain. For the simulation purposes, only one such solution is necessary. Hence, in the literature, there are several algorithms that describe how to construct a single solution having certain properties. We will discuss below the

DND algorithm introduced in [45] for the FPGA simulations and further considered for CUDA hardware.

Another interesting possibility is to consider the inequalities (1) as constraints in an ILP. This allows for faster solving algorithms, however it is important to specify the objective function that will chose one of the solutions. For example, using $\max x_1$ as objective function with the constraints from Example 1, 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).

It is remarked that the algorithm from [67] uses a different approach. It supposes that for a P system Π working in the derivation mode δ there exists a function $NBVariants(\Pi, C, \delta)$ that for any configuration C gives the number of solutions of (1). Next, it also supposes that there exists a 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).

We explain briefly the functioning of the DND algorithm in the view of equations (1). First a random rule permutation is computed. 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 validation of inequalities (1b). The computation of the random permutation and of the initial random rule frequency can be written in the form of an objective function. The first property corresponds to a probabilistic choice of the carefully chosen weights of the sum of all variables x_i to be placed in the objective function, while the second one gives constraints on the number of objects to be placed. However, it turns out that it is quite tedious to specify the corresponding objective function, which requires carefully assign the weights to the variables (by solving an additional system of equations).

F An Overview of Existing FPGA Simulations

With the advent of reconfigurable hardware which realizes the idea of modifying the hardware circuits by programming, conceiving a novel circuit simulating an innovative processing paradigm is no longer a exceedingly hard task. The first attempts to use FPGA reconfigurable hardware to simulate P systems date back to 2003 [50]. Since then two simulation approaches emerged, considering the region or the rules as basic processing units.

F.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.

As the two primary features of intuitive conceptual understanding of a P system, membranes and regions partitioned by membranes are not directly implemented as hardware circuit modules. The rule-oriented implementation approach explicitly represent the evolution rules as processing units and multisets of objects as register arrays, while implicitly represent membranes and regions as logical constructions existing between those processing units and data structures. The absence of membranes impairs the understandability according to the intuitive conceptual understanding of a P system and the membrane-mediated behaviors cannot be taken into consideration. The motivation for transforming the rule-oriented design to the region-oriented way is focus on providing a framework to incorporate important intuitive features such as cell-to-cell connections and membrane-mediated rules so as to heighten the extensibility of the hardware platform. Another intention of region-oriented approach is to distribute computational activities in P systems. This distribution fits the comprehension of intuitive concept of a P system and can vary the amount of hardware resource with the size of the P system to be implemented. The region-oriented approach contributes to constitute larger systems with different scales of P systems.

Petreska and Teuscher simulation [50] As the spearhead of simulating membrane computing on FPGA, some groundbreaking matters had been devised by Petreska and Teuscher, for instance, trading communication for membrane containment relations, taking the priorities of rules into account, proposing the first attempt to simulate membrane creation and dissolving mechanism in integrated circuit, etc. Their outstanding achievement inspired the successors to engage in this challenging and breathtaking field to advance the development of hardware simulation of P systems. For the sake precision the general model of a P system is modified in two aspects: the application of evolution rules in each membrane is not done in a maximally parallel but in a sequential manner (still keeping a parallelism at the system level); the non-deterministic evolution of configuration is substituted by a definite transition following a predetermined order. This corresponds to an ILP with the objective function as a weighted sum of variables with predefined fixed weights.

In theory, membranes are borders without internal structures and material consistence. The primary functionality of membranes is breaking new grounds for computations and their size and their placement are not important. In this simulation, the membranes structures are replaced by the enclosed substances, i.e., the multisets of objects, evolution rules and children membrane architectures. The objects exchange among parent membranes and children membranes is a kind of bi-directional traversing behavior, which is the sufficient and unnecessary conditions for the membranes' containment relationships. Namely, the existing of containment relation of membranes does not always means that there are objects interchanges, not vice versa. The widely existing of bi-directional transfer for objects are handled as communications. In case of the possible objects exchange invoked in after steps, the communications realized by data bus connecting to different parts of hardware are prearranged in all containment cases. The interconnections are in direct proportion to children membranes. To avoid the multiple buses used to connect the upper-immediate membrane to its plural lower-immediate membranes, a bus links all the children membranes before it connects to the upper-immediate membrane. The communication only presents between upper-immediate membrane and upper-immediate membranes, its children membranes. There are no objects exchange among children membranes or non-immediate contained membranes. In general, a membrane of P system corresponding to an area of integrated circuits storing objects specifying multisets of objects and sets of rules. The containment relation of immediate-include membranes is substituted by a bus connecting them.

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 the vector of numbers indicating the multiplicity of each object. The order of these registers is in accordance with the lexicographic order of the alphabet of objects. The recognizing 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 rules' left-hand side and right-hand side into different registers separately. The fact that rules are not the processing unit or something like that makes the simulation becoming a region-based implementation. 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 concurrently, which is different to 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 is stored in corresponding registers. The 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 terminates 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.

The simulation 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, for the hardware reconfiguration will cause the reconstruction of buses that connect different regions of the circuits. This scheme gives rise to a disadvantage that the hardware resources cannot be released. The hardware implementation of creating 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 a membrane creating rules invoke them.

Nguyen simulation [47] In this simulation, a parallel computing platform simulating membrane computing based on FPGA named Reconfig-P is developed [43, 48]. Reconfig-P is fabricated on the basis of the region-oriented idea that regions worked as the computational entities communicating objects through message passing. The functionality of these regions is exhibited by the set of evolution rules included. This transformation of the mentality is in line with the intuitive conceptual understanding of a P system and the extensibility of the Reconfig-P is enhanced. It can also simulate P systems in software to test and to verify the planned evolutions configuration-by-configuration, then generates hardware circuits when needed. 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 source codes. Software simulation of the circuits to be constructed in light of hardware source codes is supported by P Builder to test the functionality of circuits before mapping the codes to hardware circuits.

The execution of a evolution step is divided into two phases: *object assignment phase* and *object production phase*. 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 status is interpreted as temporal order which the region processing units should respect in the assignment phase. Specifically, region processing units process rules in line with priorities (expressed as temporal order), rules with same priority are executed concurrently. The temporal order is determined at compile-time. It should be noted that the region-oriented hardware implementation is designed for the general case, there is no derivation mode to restrict the evolution. What applied is just a rule not a multiset of applicable rules. The 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 immediate after a applicable is applied to consume some objects.

The membrane traversing behavior of objects 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, two solution strategies, the *space-oriented strategy* and the *time-oriented strategy* are proposed. For the space-oriented strategy, the register storing the object causing the conflict is replicated to the same number of parallel process so that the respective processing unit updates the value in the assigned copy register. For the time-oriented strategy, time delays are interleaved among the conflicting parallel processes so that the updating is performed in different time. In fact in the rule-oriented design which will be detailed hereinafter, the solution for the conflict is basically the same. The object production phase is completed in two clock cycles when the space-oriented strategy is adopted. In the time-oriented strategy, considering the traversing behavior of objects, the objects causing conflict are partitioned as *internal objects* and *external objects*. The internal objects refer to the objects generated by the rules without traversing behaviors, while the external objects refer to the opposite. The amount of interleaving inserted to the updating process caused by the internal objects is conformed at compile-time. The interleaving caused by the receiving of external objects is determined at run-time for the region processing units work independently. They cannot aware of the future transferring of objects. An method involves the semaphores is adopted to determine the appropriate number of interleaving during the run-time.

To avoid the disorder of two phases and execute operations in parallel, the synchronization of object assignment phase is indispensable. This synchronization is implicitly achieved by the communication of different region processing units on channels. To observe the evolution transition-by-transition, a *system coordination processing unit* is designed to coordinate the processing of region processing units. Each region processing unit gets connected to the system coordination processing unit by specialized channels. When the system coordination processing unit receives signals indicating the accomplishment of tasks from every region processing unit, the configuration evolves to next one. It might also be noted that when a couple of processing units are communicating with each other by passing objects, they cannot do any other jobs since the communication channels work in a synchronous mode. In consequence, the communications among different processing units must be arranged advance to prevent the deadlock caused by the synchronous mode. In fact, the communication behaviors are executed in different parallel branches to avoid the deadlock.

The extensibility of the region-oriented design is promoted on the basis of representing membranes as processing units interact with two region processing units corresponding to inner and outer regions. A composition of the three processing units can implement antiport rules. Particularly, the processing units stand for the inner and outer regions send objects to the membrane processing unit. Once the membrane processing unit makes a pair according to the definition of the antiport rule, it will send the coupled objects to the corresponding regions.

F.2 Rule-based simulations

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

Nguyen simulation [44] [46] 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. However, a processing unit does not correspond a concrete hardware component but corresponding to a potential infinite *while loop* which includes codes related to the applying of the rules in Handel-C code. The tags of information associated to execution and synchronization are 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 containment 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 that located in the same region with the rule processing unit. If there are objects traversing rules which imply the containment, connecting the rule processing unit to the object array to which the rule will send objects contained in different region. By this measure, the containment is realized.

As the theoretical exhibition of parallelism, different rules can and must consume and produce the same kind of object simultaneously where necessary. However, for the hardware implementation, conflicts occur when different processing units in a same region alter the multiplicities of the same objects at the same time. This fact prevents the realization of theoretical parallelism. To handle this collision, the operation of executing a rule is split into preparation phase and updating phase. In the preparation phase, each processing unit calculates the maximal number of instance for a rule with a division operation that divides the multiplicity of each objects by the number of corresponding object defined in left-hand side of a rule. A serial of quotients will obtain. Apparently, the minimum of the quotients is the maximum number of instance of a rule in terms of the Buckets Effect. In fact, the computations of maximal number of instances for rules are performed in an order with respect to the relative priorities among rules. In detail, counting the quantity of clock cycles consumed for computing the maximal instance number of rules with higher priorities at first. After that, an number of delay statements that equivalent to the number of clock cycles consumed previously are interposed above the statements for computing the maximum instance number of rules with lower priorities in the Handel-C codes. For rules with the same priority, this calculation is executed in parallel.

In the updating phase, if the maximum number of instance of a certain rule is larger than zero, then the rule is applicable. As stated before, when different processing units updates the same multiplicity value of objects registers at the same time, the conflicts arise. In order to resolve the conflict, P Builder construct a conflict matrix to detect the possibility of conflicts firstly. A row of the matrix is a four-tuple (p, q, r, s) whose p stands for an object, q is a region of the P system, r represents the set of rules gives rise to the generation or consumption of p in q, i.e. the set of rules which conflicts. s is the conflict degree of (p, q), referring to the number of conflict rules included in set r. P Builder prevents every processing unit from writing to the same register simultaneously on the basis of the analyzing the conflict matrix. Likewise, the *space-oriented strategy* and the *time-oriented strategy* are adopted to handle the conflict.

The synchronization is executed by three flags contained in the rule processing units. The region-level parallelism means that applicable rules in a region can be executed in parallel and system-level parallelism implies that performs the region-level parallelism concurrently. The simulation presented achieves region-level and system-level parallelism, while the non-determinism and other features of membrane computing are left out.

Verlan and Quiros simulation [67] [54] From the most abstract point of view, a P system can be regarded as a multiset rewriting system that objects and rules are contained by a skin membrane. This abstraction is accomplished by encoding children membrane structures as objects. The introduction of the notion multiset rewriting system can simplifies the membrane architectures. As prevailed in membrane computing, the evolving of configuration is accomplished by applying applicable rules simultaneously and non-deterministically. The successive transitions of configurations define the computing process of P systems. Because there is only one skin membrane, the *solution* collapses to *complete solution* and *dynamic solution* collapses to a *feasible solution*. In this simulation, the *complete solution* defined in Section D.1 is in the form of $Applicable(\Pi, C, \delta)$, where Π denotes the P system, C the configuration and δ indicates the derivation mode.

The target model is a static P system whose structure does not change during the computation process. Since the P system is regarded as multisets rewriting system with one skin membrane, the static P system does not embody inner membranes. Namely, there is only one region. One complete solution corresponds to one region under a certain configuration. The validity of a complete solution persists during the "life time" of one configuration, once a randomly selected solution is applied to evolve the present configuration to next one, the former complete solution is no longer in force and the new complete solution generated implicitly. As remarked above, computing the complete solution should be avoided. An elegant strategy was elaborated in order not to 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.

Devising an algorithm which carries out the computation of the cardinality and of the specific element of the complete solution in constant time on FPGA is the key issue of the approach. A remarkable characteristic of FPGA is that if 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 of the feasible solution is accomplished by two functions hardwired into the circuit: $NBVariants(\Pi, C, \delta)$, which gives the number of solutions for the configuration C and $Variant(n, \Pi, C, \delta)$, which returns the n*-th* element of $Applicable(\Pi, C, \delta)$.

A concept named rules' dependency graph is introduced to compute the two functions above. The picture below depicts the rules' dependency graph for rules $r_1 : ab \to u$ and $r_2 : bc \to v$. Assume that the derivation mode is maximal parallelism (*max*).

$$\begin{tikzcd}
 & r_1 \arrow[dl, dash] \arrow[dr, dash] & & r_2 \arrow[dl, dash] \arrow[dr, dash] & \\
a & & b & & c
\end{tikzcd}$$

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

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

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

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

To verify the conclusion, 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 $Applicable(\mathit{\Pi}, C, max)$ as bellow:

$$Applicable(\mathit{\Pi}, C, max)_1 = r_1^{3+2} r_2^{0+0} = r_1^5, p = 3, q = 0$$

$$Applicable(\mathit{\Pi}, C, max)_2 = r_1^{2+2} r_2^{1+0} = r_1^4 r_2, p = 2, q = 1$$

$$Applicable(\mathit{\Pi}, C, max)_3 = r_1^{1+2} r_2^{2+0} = r_1^3 r_2^2, p = 1, q = 2$$

$$Applicable(\mathit{\Pi}, C, max)_4 = r_1^{0+2} r_2^{3+0} = r_1^2 r_2^3, p = 0, q = 3.$$

Apparently, the results are in line with the conclusions. The same results can be easily obtained using formal powers associated to context-free languages. In this case, the language $L_N = r_1^p r_2^q \mid p + q = N$ is regular and the generation function for L_N is $q_0 = 1/(1-x)^2$. By expanding q_0 to acquire the *n-th* coefficient we obtain that $[x^n]q_0 = n + 1$, which is the function to compute $NBVariants(\mathit{\Pi}, C, max) = N + 1$. The $Variant(n, \mathit{\Pi}, c, max)$ 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 automata execution.

Communications among different processing units do harmful to improve the computing speed. For pursuing a better speed performance, modularity is adopted to minimize the interconnections of configurable logic blocks. A layer structure which just communicates with previous one is constructed to execute the algorithm. Have in mind that P system is abstracted as a multisets rewriting system with a skin membrane, the compartment is simplified. In consequence, the rules and multisets of objects are the key materials for the implementation. As usual processing scheme for multisets of objects, registers are employed to store them in terms of configurations. While for the treatment of rules in this implementation, there is no explicit mapping from rules to hardware components, on account of the fact that not a single rule, but the dependency graph of rules is fabricated for the construction of $Applicable(\mathit{\Pi}, C, max)$ which can be represented as a regular language. The entire process of the implementation is split into several consecutive stages detailed bellow, which take charge of different operations associated to phases of evolutions of configurations.

Persistence stage stores the states that the hardware system goes through. A independent stage computes the maximal instance of each rule by means of the dividing operation and MIN logic operation. *Assignment* stage 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.

The hardware system is separated into six blocks detailed as follows. *controlBlock* takes charge of supplying communications and control actions, including logic related to halting conditions. Given a configuration, if $Applicable(\mathit{\Pi}, C, max)$ is empty, or the configuration stops to evolve, then the system halts. *inoutBlock* links to the software which provides the communication with host computer. *persistenceBlock* is used for saving and updating the current configuration and partially examining the halting condition. *independentBlock*, which is independent of the derivation mode, receives the multisets of objects of the current configuration from *persistenceBlock* and carries out division and multiplication operations. The functionality of *assignBlock* corresponds to *assignment* stage. A maximum instances of an applicable rule is sent to this unit to computes $NBVariants(\mathit{\Pi}, C, max)$ and $Variant(n, \mathit{\Pi}, C, max)$. Each rule corresponds to a sub-block executing the logic of the automaton which recognizes the regular language in terms of the dependency graph of rules. *appBlock* executes $Variant(n, \mathit{\Pi}, C, max)$ to modify the multiplicity of objects to evolve the configuration to next one.

The implementation of the concerned P system is achieved by the last four blocks. The four blocks consume one clock cycle to execute their work except AssignBlock, which demand two clock cycles. Consequently, five clock cycles are required to compute the $NBVariants(\mathit{\Pi}, C, max)$, $Variant(n, \mathit{\Pi}, C, max)$ and to apply the generated solution. In this simulation, a new strategy implementing non-determinism is proposed. This strategy based on representing the complete solution as a regular language. The obtained performance speed is about 2×10^7 computational steps per second. As a disadvantage, this simulation is limitd to the classes of P systems whose complete solutions can be represented as regular languages.

G An Overview of Existing CUDA simulations

In [38], 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*: GPU implements a shared memory system, avoiding communication overload;
- *A medium scalability degree*: the amount of resources depend on the GPU model, e.g. a K40 includes 2880 cores and 12 GBytes of memory. If the resources of a GPU is 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.

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

G.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 [10]. This simulator performs only one computation out of the whole tree, in order 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 constrains, 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 [10], 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 require 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 simulated P system, and drops as long as the variety of different objects appearing in membranes decreases.

The performance of the simulator was analysed on a GPU Tesla C1060 (240 cores, 4GB memory) by using two benchmarks [38]: a simple test P system designed to stress the simulator (up to 7x of speedup), and a family of P systems solving the SAT problem (1.67x of speedup).

Improvements to this design have followed [38], reporting up to 38x 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 solving SAT in linear time [11]. In this way, the worst case assumed before (all objects defined can appear in every membrane) is further reduced by analysing 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: a thread block is assigned to each elementary membrane, and each thread to each object of the input multiset. The experiments carried out on a NVIDIA Tesla C1060 GPU reported up to 63x 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 [11].

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 with cell division was simulated [38]. The design is similar to the one for cell-like models: each thread block is assigned to each cell, but the number of objects to be placed inside each cell in the memory representation is increased, respecting to the solution in active membranes. On the contrary, this simulator does not need to store nor handle charges associated to membranes. Experiments on a NVIDIA Tesla C1060 GPU leaded to speedup by 10x. 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.

G.2 Population Dynamics P systems

Population Dynamics P (PDP) systems are multienvironment, where in each environment, a cell-like P system is placed. They have been successfully used as a modelling framework for real ecosystems. Thus, their efficient simulation is critical for virtual experimentation and experimental validation. Simulators for PDP systems typically run several simulations in order to extract statistical information from the models.

A CUDA simulator for PDP systems was presented in [39]. The selected simulation algorithm was the DCBA [40], 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 in order 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.

The CUDA simulator for PDP systems [38, 39] uses a design based on the following: environments and simulations are distributed through thread blocks, and rule blocks among 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 a first benchmark using a set of randomly generated PDP systems (without biological meaning), speedups of up to 7x were achieved on a Tesla C1060 with respect to a multi-core 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 4.9x with a C1060, and 18.1x using a Tesla K40 GPU (2880 cores).

G.3 Spiking Neural P systems

Parallel simulation of Spiking Neural P (SNP) systems has been based on a matrix representation so far [71]. 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.
- 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 spiking vector.
- Configuration vector: defines the number of spikes per neuron, that is, the configuration in a given time.

CuSNP is a simulator for SNP systems which 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 in parallel, leading to speedups of up to 2.31x [38]. Further extensions have followed, enabling delays, support of more types of regular expressions and input of P-Lingua files [9]. These leaded to up 50x of acceleration on a GTX750 GPU.

Furthermore, the simulation of Fuzzy Reasoning Spiking Neural (FRSN) P systems on the GPU was explored [31]. 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.

G.4 Other models

Enzymatic Numerical P systems has been employed for modelling robot controllers, being significant for the Artificial Intelligence. A CUDA simulator was developed [38], 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 11x.

Evolution-Communication P systems with Energy (ECPE) [38] 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.

H Other Approaches

Besides FPGA based and CUDA-enabled GPU based hardware platforms developed for the implementation (or simulation, the two terms are equivalent in the context) of P systems, the micro-controller is another hardware device which is taken into account by the researchers. A serial algorithms and their hardware circuits designs in terms of the *exhaustive investigation line* focusing on implementing the transition of configurations of P systems are developed. The nascent researches do not confine to concrete hardware devices, just designing the circuits aiming at simulating certain operations of particular P systems with registers, logical gates, magnitude comparators, and data buses.

In [19], a digital circuit is presented to select *active* rules in the current configuration. Each evolution rule is represented by 2 hardware registers. The first register characterizes the left-hand side (antecedent) of a rule, and the other specify the right-hand side of the rule, which determines whether the rewritten objects go out from the current membrane or stay where they are, or go in to 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. In succession, an algorithm computing the number of the application of active rules given the multiset of objects and evolution rules [34]. The corresponding circuit is composed of registers, logical gates, multiplexer and sequential elements. The computation process is bounded. In [27], a P system circuit is constructed by means of a micro-processor PIC16F88 plus the storage component 24LC1025, which connected by a I2C bus. The shortcoming of insufficient storage capacity of micro-processor is overcame by the introduced external memory. The flexibility of the circuit is acceptable for modifying the structure is not necessary. The circuit is depicted in Figure 7.

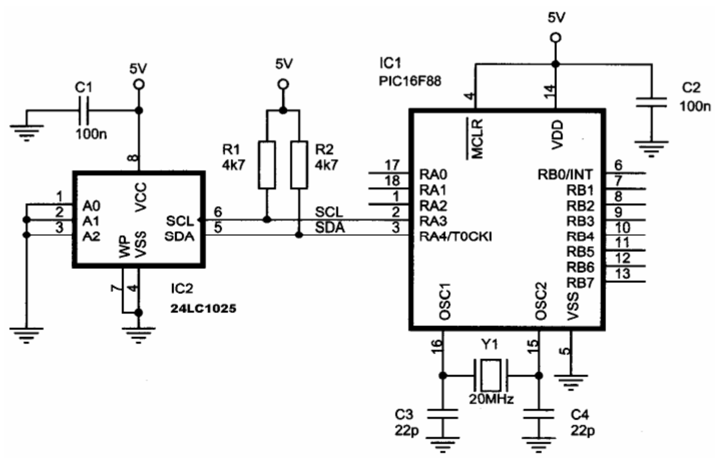


Fig. 7: A micro-processor based P system circuit from [27].

As the further research of [34], in [35], the improved algorithm and its circuit calculating the application times of active rules are investigated. The computing process can be complete in minor steps and the theoretical performance is optimized. In [3], a circuit tries to implement the inherent parallelism of P system is drafted. Towards the parallelism, the treatment to rules is similar with the thought *regional solution* defined in section 4, namely, what applied is a multiset of rules instead of a single rule. An operating environment is elaborated in [8] which performs the automatic transforming 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 to a transparent form. What should be emphasized here is that all the hardware circuits introduced are just on the blueprints which do not put into practice. They are theoretical analysis, the actual functionality and performances are unknown, unlike the FPGA based and CUDA based hardware implementations/simulations which are carried out practically.

As a new attempt for implementing neural P systems on different hardware, DRAM-based CMOS circuits are adopted to construct elementary Spiking neural P systems, which had not been implemented on FPGA but on CUDA hardware. We do not carry out an in-depth discussion about this topic given the length of the article, interested people can refer to [70] for more details.

I Conclusions

In this article we gave an overview of different hardware implementations of P systems. The FPGA approach is very promising, yielding truly parallel implementations, but 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 the programming in such languages tend to be extremely tedious.

By contrast, CUDA 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 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 is given by the construction from Section E, 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 use existing solvers and algorithms to quickly obtain the desired solution.

Finally, as a future research interest we mention the implementation of different features of P systems like membrane creation/dissolution, as well as non-classical variants of P systems like numerical P systems.

References

1. Nvidia cuda programming guide.
2. O. Agrigoroaiei, G. Ciobanu, and A. Resios. Evolving by maximizing the number of rules: Complexity study. In Membrane Computing: 10th International Workshop, WMC 2009, Curtea de Arges, Romania, August 24-27, 2009, pages 149–157, Berlin, Heidelberg, 2010. Springer.
3. S. Alonso, L. Fernandez, F. Arroyo, and J. Gil. A circuit implementing massive parallelism in transition P systems. Information Technologies and Knowledge, 2:35–42, 2008.
4. S. Alonso, L. Fernandez, F. Arroyo, and J. Gil. Main modules design for a HW implementation of massive parallelism in transition p-systems. Artificial Life and Robotics, 13(1):107–111, 2008.
5. A. Arteta, L. Fernandez, and J. Gil. Algorithm for application of evolution rules based on linear diofantic equations. In V. Negru, T. Jebelean, D. Petcu, and D. Zaharie, editors, SYNASC 2008, 10th International Symposium on Symbolic and Numeric Algorithms for Scienti
c Computing, Timisoara, Romania, 26-29 September 2008, pages 496–500. IEEE Computer Society, 2008.
6. A. V. Baranda, F. Arroyo, J. Castellanos, and R. Gonzalo. Towards an electronic implementation of membrane computing: A formal description of non-deterministic evolution in transition P systems. In N. Jonoska and N. C. Seeman, editors, DNA Computing, 7th International Workshop on DNA-Based Computers, DNA7, Tampa, Florida, USA, June 10-13, 2001, Revised Papers, volume 2340 of Lecture Notes in Computer Science, pages 350–359. Springer, 2001.

7. G. Bravo, L. Fernndez, F. Arroyo, and J. Frutos. A hierarchical architecture with parallel comunication for implementing P systems. Information Technologies and Knowledge, 2(1):43–48, 2008.
8. S. M. G. Canaval, A. G. Rodriguez, and S. A. Villaverde. Hardware implementation of P systems using microcontrollers. an operating environment for implementing a partially parallel distributed architecture. In Symbolic and Numeric Algorithms for Scienti
c Computing, 2008. SYNASC'08. 10th International Symposium on, pages 489–495. IEEE, 2008.
9. J. P. A. Carandang, J. M. B. Villaores, F. G. C. Cabarrle, H. N. Adorna, and M. A. Martnez-del-Amor. Cusnp: Spiking neural P systems simulators in CUDA. Romanian Journal of Information Science and Technology, 20(1):57–70, 2017.
10. J. M. Cecilia, J. M. Garcia, G. D. Guerrero, M. A. Martnez-del-Amor, I. Perez-Hurtado, and M. J. Perez-Jimenez. Simulation of P systems with active membranes on CUDA. Briefings in Bioinformatics, 11(3): 313–322, 2010.
11. J. M. Cecilia, J. M. Garcia, G. D. Guerrero, M. A. Martnez-del-Amor, M. J. Perez-Jimenez, and M. Ujaldon. The GPU on the simulation of cellular computing models. Soft Comput., 16(2):231–246, 2012.
12. G. Ciobanu, S. Marcus, and G. Paun. New strategies of using the rules of a P system in a maximal way: Power and complexity. Romanian Journal of Information Science and Technology, 12(2):157–173, 2009.
13. G. Ciobanu and A. Resios. Complexity of evolution in maximum cooperative P systems. Natural Computing, 8(4):807, Jan 2009.
14. G. Ciobanu and G. Wenyuan. P systems running on a cluster of computers. In International Workshop on Membrane Computing, pages 123–139. Springer, 2003.
15. M. S. Dodd, D. Papineau, T. Grenne, J. F. Slack, M. Rittner, F. Pirajno, J. ONeil, and C. T. Little. Evidence for early life in earths oldest hydrothermal vent precipitates. Nature, 543(7643):60–64, 2017.
16. R. U. F. Varela, H. Maturana. Autopoiesis: The organization of living systems, its characterization and a model. BioSystems, 5:187–196, 1974.
17. L. Fernandez, F. Arroyo, J. Castellanos, J. A. Tejedor, and I. Garcia. New algorithms for application of evolution rules based on applicability benchmarks. In Proceedings of the 2006 International Conference on Bioinformatics & Computational Biology, BIOCOMP'06, Las Vegas, Nevada, USA, June 26–29, 2006, pages 94–102.
18. L. Fernandez, F. Arroyo, I. Garcia, and G. Bravo. Decision trees for applicability of evolution rules in transition P systems. Information Theories and Applications, 14(3):223–230, 2007.
19. L. Fernandez, V. J. Martnez, F. Arroyo, and L. F. Mingo. A hardware circuit for selecting active rules in transition P systems. In 7th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, September 25-29, 2005, Timisoara, Romania, pages 415–418.
20. L. Fernando, F. Arroyo, J. Tejedor, and J. Castellanos. Massively parallel algorithm for evolution rules application in transition P systems. In Proceedings of 7th Workshop on Membrane Computing, pages, 337–343. Lorentz center, Univeristeit Leiden, 2006.
21. R. Freund, A. Leporati, G. Mauri, A. E. Porreca, S. Verlan, and C. Zandron. Flattening in (tissue) P systems. In 14th International Conference on Membrane Computing, Chisinau, Republic of Moldova, August 20-23, 2013, Lecture Notes in Computer Science, volume 8340, pages, 173–188.
22. R. Freund, I. Perez-Hurtado, A. Riscos-Nunez, and S. Verlan. A formalization of membrane systems with dynamically evolving structures. International Journal of Computer Mathematics, 90(4): 801-815, 2013.

23. R. Freund and S. Verlan. A formal framework for static (tissue) P systems. In G. Eleftherakis, P. Kefalas, G. Paun, G. Rozenberg, and A. Salomaa, editors, Mem- brane Computing, 8th International Workshop, WMC 2007, Thessaloniki, Greece, June 25-28, 2007 Revised Selected and Invited Papers, volume 4860 of Lecture Notes in Computer Science, pages 271–284. Springer, 2007.
24. A. Gutierrez, L. Fernandez, F. Arroyo, and S. Alonso. Hardware and software architecture for implementing membrane systems: A case of study to transition P systems. In 13th International Meeting on DNA Computing, Memphis, TN, USA, June 4-8, 2007, volume 4848 of Lecture Notes in Computer Science, pages 211–220. Springer, 2007.
25. A. Gutierrez, L. Fernandez, F. Arroyo, and S. Alonso. Suitability of using micro-controllers in implementing new P-system communications architectures. Artificial Life and Robotics, 13(1):102–106, Dec 2008.
26. A. Gutierrez, L. Fernandez, F. Arroyo, and G. Bravo. Optimizing membrane system implementation with multisets and evolution rules compression. In Proceedings of the 8th Workshop on Membrane Computing, pages 345–362, 2007.
27. A. Gutierrez, L. Fernandez, F. Arroyo, and V. J. Martnez. Design of a hardware architecture based on micro-controllers for the implementation of membrane systems. In 8th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, 26-29 September 2006, Timisoara, Romania, pages 350–353. IEEE Computer Society, 2006.
28. E. K. Jason Sanders, editor. CUDA by Example:An IntroductIon to General- Purpose GPU ProgrammIng. Addison Wesley, 2010.
29. F. Javier Gil, L. Fernndez, F. Arroyo, and J. Tejedor. Delimited massively parallel algorithm based on rules elimination for application of active rules in transition P systems. Information Technologies and Knowledge, 2(1):56–61, 2008.
30. D. Kirk and W.-M. Hwu. Programming Massively Parallel Processors: A Hands On Approach. Morgan Kaufmann.
31. L. F. Macas-Ramos, M. A. Martnez-del-Amor, and M. J. Perez-Jimenez. Simulating FRSN P systems with real numbers in p-lingua on sequential and CUDA platforms. In Membrane Computing - 16th International Conference, Valencia, Spain, August 17-21, 2015, Revised Selected Papers, pages 262–276, 2015.
32. M. Madhu, V. S. Murty, and K. Krithivasan. A hardware realization of p systems with carriers. In Poster presentation at the Eight International Conference on DNA based Computers, Hokkaido University, Sapporo Campus, Japan, 2002.
33. V. Martinez, S. Alonso, and A. Gutierrez. Hardware circuit for the application of evolution rules in a transition P-system. Artificial Life and Robotics, 15(1):89–92, Aug 2010.
34. V. Martinez, F. Arroyo, A. Gutierrez, and L. Fernandez. Hardware implementation of a bounded algorithm for application of rules in a transition P-system. In Symbolic and Numeric Algorithms for Scienti
c Computing, 2006. SYNASC'06. Eighth International Symposium on, pages 343–349. IEEE, 2006.
35. V. Martinez, L. Fernandez, F. Arroyo, and A. Gutierrez. HW implementation of a optimized algorithm for the application of active rules in a transition P-system. Information Theories and Applications, 14(4):324–331, 2007.
36. V. Martnez, A. Gutierrez, and L. F. Mingo. Circuit FPGA for Active Rules Selection in a Transition P System Region, pages 893–900. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.
37. V. J. Martnez, F. Arroyo, A. Gutierrez, and L. Fernandez. Hardware implementation of a bounded algorithm for application of rules in a transition P-system. In V. Negru, D. Petcu, D. Zaharie, A. Abraham, B. Buchberger, A. Cicortas, D. Gorgan, and J. Quinqueton, editors, 8th International Symposium on Symbolic and Numeric Algorithms for Scienti

c Computing (SYNASC 2006), 26-29 September 2006, Timisoara, Romania, pages 343–349. IEEE Computer Society, 2006.
38. M. A. Martnez-del-Amor, M. Garca-Quismondo, L. F. Macas-Ramos, L. Valencia-Cabrera, A. Riscos-Nunez, and M. J. Pere-Jimenez. Simulating P systems on GPU devices: A survey. Fundam. Inform., 136(3):269–284, 2015.
39. M. A. Martnez-del-Amor, L. F. Macas-Ramos, L. Valencia-Cabrera, and M. J. Perez-Jimenez. Parallel simulation of population dynamics P systems: updates and roadmap. Natural Computing, 15(4):565–573, 2016.
40. M. A. Martnez-del-Amor, I. Perez-Hurtado, M. Gatca-Quismondo, L. F. Macas-Ramos, L. Valencia-Cabrera, A. R. Jimenez, C. G. Daz, A. Ricsoc-Nunez, M. A. Colomer, and M. J. Perez-Jimenez. DCBA: simulating population dynamics P systems with proportional object distribution. In Membrane Computing - 13th International Conference, CMC 2012, Budapest, Hungary, August 28-31, 2012, Revised Selected Papers, pages 257-276, 2012.
41. M. A. Martnez-del-Amor, I. Perez-Hurtado, M. J. Perez-Jimenez, A. Riscos-Nunez, and M. A. Colomer. A new simulation algorithm for multienvironment probabilistic P systems. In Fifth International Conference on Bio-Inspired Computing: Theories and Applications, BIC-TA 2010, University of Hunan, Liverpool Hope University, Liverpool, United Kingdom / Changsha, China, September 8-10 and September 23-26, 2010, pages 59–68, 2010.
42. C. Max
eld, editor. FPGAs World Class Designs. Elsevier, 2009.
43. V. Nguyen. An Implementation of the Parallelism, Distribution and Nondeter- minism of Membrane Computing Models on Recon
gurable Hardware. PhD thesis, University of South Australia, 2010.
44. V. Nguyen, D. Kearney, and G. Gioiosa. Balancing performance, exibility, and scalability in a parallel computing platform for membrane computing applications. In G. Eleftherakis, P. Kefalas, G. Paun, G. Rozenberg, and A. Salomaa, editors, Membrane Computing, 8th International Workshop, WMC 2007, Thessa- loniki, Greece, June 25-28, 2007 Revised Selected and Invited Papers, volume 4860 of Lecture Notes in Computer Science, pages 385–413. Springer, 2007.
45. V. Nguyen, D. Kearney, and G. Gioiosa. An algorithm for non-deterministic object distribution in P systems and its implementation in hardware. In D. W. Corne, P. Frisco, G. Paun, G. Rozenberg, and A. Salomaa, editors, Membrane Comput- ing - 9th International Workshop, WMC 2008, Edinburgh, UK, July 28-31, 2008, Revised Selected and Invited Papers, volume 5391 of Lecture Notes in Computer Science, pages 325–354. Springer, 2008.
46. V. Nguyen, D. Kearney, and G. Gioiosa. An implementation of membrane computing using recon
gurable hardware. Computing and Informatics, 27(3+):551–569, 2008.
47. V. Nguyen, D. Kearney, and G. Gioiosa. A region-oriented hardware implementation for membrane computing applications. In 10th International Workshop on Membrane Computing, WMC 2009, Curtea de Arges, Romania, August 24-27, 2009. Revised Selected and Invited Papers, volume 5957 of Lecture Notes in Computer Science, pages 385–409. Springer, 2009.
48. V. Nguyen, D. Kearney, and G. Gioiosa. An extensible, maintainable and elegant approach to hardware source code generation in recon
g-p. J. Log. Algebr. Program., 79(6):383–396, 2010.
49. T. Y. Nishida. A membrane computing model of photosynthesis. Applications of Membrane Computing, pages 181–202, 2006.
50. B. Petreska and C. Teuscher. A recon
gurable hardware membrane system. In Membrane Computing, International Workshop, WMC 2003, Tarragona, Spain, July 17-22, 2003, Revised Papers, volume 2933 of Lecture Notes in Computer Science, pages 269–285. Springer, 2003.

51. A. Pohorille and D. Deamer. Self-assembly and function of primitive cell membranes. Research in microbiology, 160(7): 449–456, 2009.
52. G. Paun, editor. Menbrane Computing: An Introduction. Springer, 2002.
53. G. Paun, G. Rozenberg, and A. Salomaa, editors. The Oxford Handbook of Membrane Computing. Oxford University Press, 2009.
54. J. Quiros, S. Verlan, J. Viejo, A. Millan, and M. J. Bellido. Fast hardware implementations of static P systems. Computing and Informatics, 35(3):687–718, 2016.
55. R. Reina-Molina, D. Daz-Pernil, and M. Gutirrez-Naranjo. Integer linear programming for tissue-like P systems. In Proceedings of the Ninth Brainstorming Week on Membrane Computing, 2011.
56. G. Rozenberg and A. Salomaa, editors. Handbook of Formal Languages, volume 1–3. Springer, 1997.
57. F. J. G. Rubio, J. A. T. Cerbel, and L. F. Mu noz. Fast linear algorithm for active rules application in transition P systems. In K. Markov, K. Ivanova, and I. Mitov, editors, Algorithmic and Mathematical Foundations of the Arti
cial Intel- ligence, volume Supple of International Book Series ?INFORMATION SCIENCE & COMPUTING?, pages 35–44, So
a, Bulgaria, June 2008. Institute of Information Theories and Applications FOI ITHEA, Bulgaria.
58. F. J. G. Rubio, L. F. Mu noz, F. A. Montoro, and J. A. T. Cerbel. Delimited massively parallel algorithm based on rules elimination for application of active rules in transition P systems. In K. Markov and K. Ivanova, editors, Proceedings of the Fifth International Conference on Information Research and Applications, i.TECH 2007, volume Vol. 1, pages 182–188, Bulgaria, June 2007. Institute of Information Theories and Applications FOI ITHEA.
59. F. J. G. Rubio, L. F. Mu noz, F. A. Montoro, and J. A. de Frutos Velasco. Parallel algorithm for P systems implementation in multiprocessors. In Proceedings of the Thirteenth International Symposium on Arti
cial Life and Robotics 2008 (AROB 13th'08), pages 0–0. M. Sugisaka and H. Tanaka, 2008.
60. Y. Suzuki, Y. Fujiwara, J. Takabayashi, and H. Tanaka. Arti
cial life applications of a class of p systems: Abstract rewriting systems on multisets. In Workshop on Membrane Computing, pages 299–346. Springer, 2000.
61. Y. Suzuki and H. Tanaka. Modeling p53 signaling pathways by using multiset processing., 2006.
62. J. A. Tejedor, L. Fernandez, F. Arroyo, and S. Gomez. Algorithm of rules applications based on competitiveness of evolution rules. In Proceedings of the 8th Workshop on Membrane Computing, pages 567–580, 2007.
63. J. A. Tejedor, L. Fernandez, F. Arroyo, and A. Gutierrez. Algorithm of active rule elimination for application of evolution rules. In Proceedings of the 8th Conference on 8th WSEAS International Conference on Evolutionary Computing, volume 8, pages 259–267, 2007.
64. J. A. Tejedor, A. Gutierrez, L. Fernandez, F. Arroyo, G. Bravo, and S. G. Canaval. Optimizing evolution rules application and communication times in membrane systems implementation. In 8th International Workshop, WMC 2007, Thessaloniki, Greece, June 25-28, 2007 Revised Selected and Invited Papers, volume 4860 of Lecture Notes in Computer Science, pages 298–319. Springer, 2007.
65. S. Verlan. Study of language-theoretic computational paradigms inspired by biology. Habilitation thesis, Universite Paris Est, 2010.
66. S. Verlan. Using the formal framework for P systems. In Membrane Computing - 14th International Conference, CMC 2013, Chisinau, Repubilc of Moldove, August 20-23, 2013, Revised Selected Papers, volume 8340 of Lecture Notes in Computer Science, pages 56–79. Springer, 2013.

67. S. Verlan and J. Quiros. Fast hardware implementations of P systems. In Membrane Computing - 13th International Conference, CMC 2012, Budapest, Hungary, August 28-31, 2012, Revised Selected Papers, volume 7762 of Lecture Notes in Computer Science, pages 404–423. Springer, 2012.
68. S. A. Villaverde, L. F. Mu noz, F. A. Montoro, and F. J. G. Rubio. A circuit implementing massive parallelism in transition P systems. International Journal Information Technologies and Knowledge., 2(1):35–42, January 2008.
69. N. Wilt, editor. The CUDA Handbook:A Comprehensive Guide to GPU Program- ming. Addison Wesley, 2013.
70. Z. Xu, M. Cavaliere, P. An, S. Vrudhula, and Y. Cao. The stochastic loss of spikes in spiking neural p systems: Design and implementation of reliable arithmetic circuits. Fundamenta Informaticae, 134(1-2):183–200, 2014.
71. X. Zeng, H. Adorna, M. A. Martnez-del-Amor, L. Pan, and M. J. Perez-Jimenez. Matrix representation of spiking neural P systems. In Membrane Computing - 11th International Conference, CMC 2010, Jena, Germany, August 24-27, 2010. Revised Selected Papers, pages 377–391, 2010.