1 Introduction

1.1 Membrane Computing Overview

Membrane computing (MC for short) is a branch of *natural computing* investigating computational models called *membrane systems* or *P systems*, inspired by the structure and functionality of the *living cell*. This computing paradigm was introduced by Gh. Păun, initially in a technical report [26] and then in a journal paper [27]. The MC research has initially developed mostly as a theoretical investigation looking at various models and bringing inspiration from a multitude of living cell concepts, topics, and phenomena. All these models have in common a set of compartments separated by membranes and organized according to a certain structure (tree, graph) that can be fixed or dynamic. Each of these compartments contains biochemical entities, called *objects*, which evolve according to local rules by transforming multisets of objects and/or moving them from a compartment to a neighboring one. This common framework uses various bioinspired features such as *activators* and *inhibitors*, *membrane electrical charges*, *catalysts*, *membrane thickness*, and cross-membrane movement of objects (*symport*, *antiport*) in order to generate various types of models, all fine-tuned by a rigorous way of selecting them in a consistent and coherent manner. The main characteristic of these models is their distributed and parallel behavior, that is, the computation takes place in each compartment and multiple transformations and/or cross-membrane movement of objects may take place in parallel. Well-defined topics for many classes of (natural computing) computational models, such as *computational power*, *complexity*, and connections with other computational models, have been investigated, revealing a wealth of new and exciting results. A first research monograph [28] included some of these initial theoretical aspects of this field.

These theoretical investigations have paved the way for applications in biology, computer science, computer graphics, and linguistics. Some tools have been also produced, supporting these research developments. All these have been published in another Springer monograph [11]. The key theoretical developments and

G. Zhang et al., *Membrane Computing Models: Implementations*,
https://doi.org/10.1007/978-981-16-1566-5_1

applications of membrane computing, at the level of 2010, have been presented in a handbook [29]. More recently, some more specific MC applications have been reported [16]. With the exploration of MC models with real-life background, more and more real-life complex and challenging applications have been investigated [14, 43].

From this very brief MC overview, it is clear that the theoretical investigations and MC applications have had a very consistent and steady development. With a multitude of models requesting, in certain circumstances, tools to verify various hypotheses and numerous applications involving the simulation, verification, and analysis of complex systems, a new type of research activity has been launched, the design and development of adequate software and hardware tools. Thus, this book will systematically report the results and applications on software and hardware implementations of a variety of MC models.

The rest of this chapter is organized as follows: Sect. 1.2 introduces software implementation of P systems. Section 1.3 introduces hardware implementation of P systems. Section 1.4 discusses challenging problems of P systems implementation. Concluding remarks, other implementations of P systems, and a brief presentation of the chapters of this book will be discussed in Sect. 1.5.

1.2 Software Implementation of P Systems

When conducting research in MC, the design of *computational models* (P systems) can be guided by a broad spectrum of questions, ranging from solving *hard* problems (i.e., problems having a high computational complexity) to modelling complex systems. The desired answer can be extracted from the analysis of the corresponding computations of the designed P system over initial scenarios of interest. Obviously, in order for the designs to be reliable, it is important to verify their correctness, that is, making sure that the described system actually reflects its expected behavior. In this context, the need for software implementations able to manipulate P systems and recreate their behavior naturally arose. Such automatic tools provide not only an invaluable help for the verification process (especially when dealing with large and complicated designs and/or long computations) but also the possibility to run virtual experiments and to process the data associated to the simulated computations (especially in the case of computational modelling of complex systems). A very brief overview of existing P systems simulators is provided in what follows. For a more detailed history and bibliography, we refer the reader to [11, 29, 33, 40].

First software implementations were written in `LISP` [38] and `Prolog` [19], followed closely by `Scheme` [2] and `Haskell` [1] simulators. Note that the term *implementation* is used in a software engineering sense (i.e., developing code that somehow captures the specification of an abstract theoretical model), but it does not correspond to a faithful and precise materialization of the semantics of the model. In particular, the inherent parallelism concerning rule applications in one step of a membrane system cannot be implemented *as is* in the sequential Central Processing

Unit (CPU) of a standard computer although it can be "imitated" or "emulated" by means of loops of intermediate auxiliary steps. Nevertheless, there were also several early attempts to get closer to such an ideal implementation by using parallel techniques of different programming languages, such as `MPI` for communicating threads in `C++` [9, 10] or `RMI` for the communication among processes in different computers in `Java` [39].

From that point on, new simulators kept coming out on a regular basis as the field grew and different research groups started to investigate new types of membrane systems. However, at that time, software tools were mostly considered as auxiliary by-products, typically featuring an *ad hoc* design oriented to a specific type of P system, and they were not meant to be extended.

In this situation, `P-Lingua` project was created [12, 13], pursuing a standard formalization that could be used by software implementations independently of the underlying programming language used to develop the simulator. The initial goal was to cover as many types of P systems as possible and to define a syntax specification, which was similar to the notation used in the MC literature. Chapter 3 offers more details about P-Lingua project, as well as about MeCoSim, a general purpose tool for virtual experimentation in membrane computing built on top of P-Lingua.

It is also worth pointing out that there exist some specialized integrated simulation tools that offer modelization services to users, which are not required to have a strong membrane computing training (mostly dealing with biochemical processes). Some of the most relevant works in this direction are *Cyto-Sim* [34], *MetaPlab / MpTheory* [6], *BioSimWare* [3], *Infobiotics Workbench* [4] (see Chap. 4), and *The Java Environment for Nature-inspired Approaches (JENA)* [17] (see Chap. 5).

1.3 Hardware Implementation of P Systems

Software simulations of P systems face difficulties in expressing in an efficient way the parallel and distributed nature of the model as current-day computers are based on a different, mostly sequential, paradigm. So from the very beginning, a research aiming to accelerate the execution of P system simulators using different types of hardware platforms was proposed. Two main research directions have been developed in this respect, depending on the underlying hardware utilized: *Graphics Processing Unit* (GPU) *Compute Unified Device Architecture* (CUDA)-based and Field Programmable Gate Array (FPGA)-based implementations. We refer to Chaps. 6 and 7 for the presentation of the corresponding hardware.

The development cycle for a GPU CUDA-based implementation is very similar to a traditional software development cycle and allows a relatively quick prototyping and implementation of corresponding algorithms. The major difficulty is to handle the data parallelism provided by these devices. In the FPGA case, a unique circuit design is created for each concrete system and its initial data. This allows to optimize the circuit for the corresponding computation and to achieve important speedups of several orders of magnitude. In order to accommodate more designs, a software generator is used that provides a hardware description corresponding to the system description and its initial configuration.

First, FPGA-based implementation of a P systems model was proposed in 2003 by Petreska and Teuscher [30], for transitional P systems and using a variant of sequential rule application strategy. Then, in 2008–2010, Nguyen et al. presented a series of implementations of transitional P systems with maximal parallelism evolution strategy [22–25]. These implementations featured up to 500 times speedup with respect to a reference software implementation. In 2012, Quiros and Verlan proposed a first truly nondeterministic implementation of a variant of network of cells (a generic P systems model; see [15]) [31, 32, 42]. This is also the first implementation achieving a speedup of order 10^4. In 2019–2020, Shang et al. proposed an implementation of numerical P systems [35–37] with several applications in robotics and achieving a speedup of order 10^5. More details about these implementations can be found in Chap. 7.

The first GPU simulators came relatively late, in 2010 [7]. However, due to a lower development effort with respect to FPGA-based designs, their number is bigger. Besides P systems with active membranes [7, 8], population dynamics P (PDP) systems [21], spiking neural P systems [5], enzymatic numerical P systems, and evolution-communication P systems with energy [20] were targeted for simulation. The obtained speedups range from 1.6 to 100 with respect to a reference software implementation.

In conclusion, we would like to remark that the development of hardware implementations is very promising as it allows to achieve important speedups. Unfortunately, the development time for these implementations, especially for FPGA-based ones, is much larger than for an ordinary software one. So there is a kind of a trade-off between the efficiency of an implementation and the development speed. Hence, for a hardware implementation to be cost-effective, it should target a problem with significant further development, such as like mobile robot controller, in order to compensate for the high development effort.

We also refer to a recent paper [44] that gives a detailed overview of different hardware implementations of P systems.

1.4 Challenges of P Systems Implementation

We mention here some of the most significant challenges of P systems implementation that are discussed in the next chapters: inherent parallelism of the MC models, a broad spectrum of models, combining in different ways various features associated with the generic framework, the usability of the tools, and nondeterminism implementation on computers with von Neumann architecture. These challenges require new algorithms, adequate software and/or hardware platforms, generic or specific solutions, and an effort, in some cases, to produce tools that might appeal to researchers outside the MC community—we have in mind those complex systems requesting models that have to be simulated and analyzed with specific tools. More details can refer to [41,44]. Some of these challenges are explained as follows:

- *Parallelism*: How to realize a P system with inherent parallelism is one of the main challenging problem in P systems software and/or hardware tools. The challenge results from both model design and simulator implementation or hardware constraints. Simulating P systems is a memory-demanding task, given that the execution of rules requires several accesses to memory for just one conditional operation [41]. The design of P system variants of high computational intensity and with memory-bandwidth bounded is a challenging task. The rule competition for objects in the selection phases and finding ways to extend the idea of adaptive simulators represent significant bottlenecks.
- *Nondeterminism*: Simulating a P system with nondeterminism on the inherently deterministic computers or hardware with von Neumann architecture is another main challenge. Pseudorandom numbers, instead pf actual random numbers, are used in the present simulators.
- *Universality and flexibility*: There are numerous P system models with their own syntactical elements, such as initial structures, initial multisets of objects and initial set of rules, and specific semantics of the execution strategy. Thus, it is a challenging task in the development of a software simulator or selection of a hardware platform that is flexible enough to support all the P system variants.

1.5 Concluding Remarks

As each of the following chapters has an abstract summarizing its content, we will not present here a summary of each chapter but will discuss instead the main problems pointing to where they appear in this book.

Three software platforms, P-Lingua (Chap. 2), Infobiotics Workbench (Chap. 4), and JENA Environment (Chap. 5), are described, pointing to their key features and usage. *P-Lingua* framework, probably the most widely used software tool, allows for the specification and simulation of a large spectrum of types of P systems. Algorithms describing the semantics of these classes of P systems and a higher level tool, called *MeCoSim*, providing a visual representation of the simulation environment, are presented. *Infobiotics Workbench* is an integrated software suite developed for computational systems biology and relying upon stochastic P systems. Its components providing computer-aided modelling and analysis of biological systems through simulation, verification, and optimization are described, and their usage illustrated with some case studies. *JENA* is a modular, configurable, and extendable platform conceived as a virtual laboratory and a virtual cell. Biological information processing is based on natural laws at a molecular level. Resulting principles make use of dedicated chemical reactions, mechanisms for transportation of biomolecules, and forces among molecules and their environment mainly induced by electric charges and by movement in local space. An introduction to JENA is presented, including its features and capabilities from the user's perspective and from a technical point of view. Four illustrative case studies are described.

Tools supporting the automatic design of various types of MC models (cell-like P systems and spiking neural P systems), by using various classes of genetic algorithms, are described in Chap. 3. The performance of the tools is proved through a set of examples. In the same chapter are illustrated the capabilities of the MeCoSim tool for modelling a complex ecosystem and issues related to the parallelization of a robot motion problem when modelled with a special class of numerical P systems.

As mentioned above, the parallelization of various processes occurring in the simulation of certain classes of MC models is a challenging aspect in building efficient tools. In this respect, implementations relying on specific hardware, namely, GPU and FPGA, are investigated in Chaps. 6 and 7, respectively. Concepts related to GPU computing and its applications are introduced. Three types of simulators are identified and presented: those developed for very specific P systems or family of P systems (specific simulators), others developed for a wide range of P systems inside a variant (generic simulators), and a hybrid simulator that receives high-level information to be better adapted (adaptive simulators). Some guidelines on how to develop new simulators for P systems on GPUs are presented.

The other hardware option for implementing parallel computation, FPGA, is discussed in the context of implementing generalized numerical P systems by considering many advanced techniques. A discussion related to the challenges posed by FPGA implementations is presented. Enzymatic numerical P systems-based robot controllers and path planning algorithm are implemented in FPGA, achieving a speedup of 10^5 and 10^4 order of magnitude compared to software simulation.

Also, there are other software tools, such as kPWorkbench [18] or MetaPlab [6], dedicated to the simulation and analysis of two specific classes of P systems, kernel P systems and Metabolic P systems, respectively, which are not presented in this book as they target topics that are not within its scope.

The gain obtained by reading this book is twofold: On the one hand, the tools presented are introduced together with a thorough investigation of various algorithms, methods, and guidelines regarding the implementation strategies, and on the other hand, a consistent description of the usage of the tools and a set of illustrative examples are presented.

We hope that the readers will find this book interesting, useful, and helpful in their own investigations and research and will open the desire to make use of these tools in modelling, analyzing, and better understanding of complex systems modelled with different types of membrane systems.

References

1. F. Arroyo, C. Luengo, A.V. Baranda, L. Mingo, A software simulation of transition P systems in Haskell, in *Membrane Computing (WMC 2002)*, ed. by Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron. Lecture Notes in Computer Science, vol. 2597 (2003), pp. 19–32. https://doi.org/10.1007/3-540-36490-0_2

2. D. Balbontín-Noval, M.J. Pérez-Jiménez, F. Sancho-Caparrini, A MzScheme implementation of transition P systems, in *Membrane Computing (WMC 2002)*, ed. by Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron. Lecture Notes in Computer Science, vol. 2597 (2003), pp. 58–73. https://doi.org/10.1007/3-540-36490-0_5
3. D. Besozzi, P. Cazzaniga, G. Mauri, D. Pescini, BioSimWare: a software for the modeling, simulation and analysis of biological systems, in *Membrane Computing (CMC 2010)*, ed. by M. Gheorghe, T. Hinze, Gh. Păun, G. Rozenberg, A. Salomaa. Lecture Notes in Computer Science, vol. 6501 (2010), pp. 119–143. https://doi.org/10.1007/978-3-642-18123-8_12
4. J. Blakes, J. Twycross, F.J. Romero-Campero, N. Krasnogor, The Infobiotics Workbench: an integrated in silico modelling platform for systems and synthetic biology. Bioinformatics **27**(23), 3323–3324 (2011). https://doi.org/10.1093/bioinformatics/btr571
5. J.P.A. Carandang, J.M.B. Villaflores, F.G.C. Cabarle, H.N. Adorna, M.A. Martínez-del-Amor, CuSNP: spiking neural P systems simulators in CUDA. Rom. J. Inf. Sci. Technol. **20**(1), 57–70 (2017)
6. A. Castellini, V. Manca, MetaPlab: a computational framework for metabolic P systems, in *Membrane Computing (WMC 2008)*, ed. by D.W. Corne, P. Frisco, Gh. Păun, G. Rozenberg, A. Salomaa. Lecture Notes in Computer Science, vol. 5391 (2008), pp. 157–168. https://doi.org/10.1007/978-3-540-95885-7_12
7. J.M. Cecilia, J.M. García, G.D. Guerrero, M.A. Martínez-del-Amor, I. Pérez-Hurtado, M.J. Pérez-Jiménez, Simulation of P systems with active membranes on CUDA. Briefings Bioinf. **11**(3), 313–322 (2010). https://doi.org/10.1093/bib/bbp064
8. J.M. Cecilia, J.M. García, G.D. Guerrero, M.A. Martínez-del-Amor, M.J. Pérez-Jiménez, M. Ujaldón, The GPU on the simulation of cellular computing models. Soft Comput. **16**(2), 231–246 (2012). https://doi.org/10.1007/s00500-011-0716-1
9. G. Ciobanu, G. Wenyuan, A parallel implementation of transition P systems, in *Pre-Proceedings of the Workshop on Membrane Computing, Tarragona, Spain, 2003*, ed. by A. Alhazov, C. Martín-Vide, Gh. Păun. Report RGML 28/03 (2003), pp. 169–184
10. G. Ciobanu, G. Wenyuan, P systems running on a cluster of computers, in *Membrane Computing (WMC 2003)*, ed. by C. Martín-Vide, G. Mauri, Gh. Păun, G. Rozenberg, A. Salomaa. Lecture Notes in Computer Science, vol.2933 (2004), pp. 123–139. https://doi.org/10.1007/978-3-540-24619-0_9
11. G. Ciobanu, M.J. Pérez-Jiménez, Gh. Păun, *Applications of Membrane Computing* (Springer, Berlin, 2005)
12. D. Díaz-Pernil, I. Pérez-Hurtado, M.J. Pérez-Jiménez, A. Riscos-Núñez, P-Lingua: a programming language for membrane computing, in *Proceedings of the Sixth Brainstorming Week on Membrane Computing*, ed. by D. Díaz-Pernil, C. Graciani, M.A. Gutiérrez-Naranjo, Gh. Păun, I. Pérez-Hurtado, A. Riscos-Núñez, Fénix Editora (2008), pp. 135–155
13. D. Díaz-Pernil, I. Pérez-Hurtado, M.J. Pérez-Jiménez, A. Riscos-Núñez, A P-Lingua programming environment for Membrane Computing, in *Membrane Computing (WMC 2008)*, ed. by D.W. Corne, P. Frisco, Gh. Păun, G. Rozenberg, A. Salomaa. Lecture Notes in Computer Science, vol. 5391 (2009), pp. 187–203. https://doi.org/10.1007/978-3-540-95885-7_14
14. S. Fan, P. Paul, T. Wu, H. Rong, G. Zhang, On applications of spiking neural P systems. Appl. Sci. **10**(20), 7011 (2020). https://doi.org/10.3390/app10207011
15. R. Freund, S. Verlan, A formal framework for static (tissue) P systems, in *Membrane Computing (WMC 2007)*, ed. by G. Eleftherakis, P. Kefalas, Gh. Păun, G. Rozenberg, A. Salomaa. Lecture Notes in Computer Science, vol. 4860 (2007), pp. 271–284. https://doi.org/10.1007/978-3-540-77312-2_17
16. P. Frisco, M. Gheorghe, M.J. Pérez-Jiménez, *Applications of Membrane Computing in Systems and Synthetic Biology* (Springer, Berlin, 2014)
17. T. Hinze, The Java Environment for Nature-inspired Approaches (JENA): A workbench for bioComputing and bioModelling enthusiasts, in *Enjoying Natural Computing, Series Lecture Notes in Computer Science*, ed. by C. Graciani, A. Riscos-Núñez, Gh. Păun, G. Rozenberg, A. Salomaa, vol. 11270 (2018), pp. 155–169. https://doi.org/10.1007/978-3-030-00265-7_13

18. S. Konur, L. Mierlă, F. Ipate, M. Gheorghe, kP-Workbench: a software suite for membrane systems. SoftwareX **11**, Article No. 100407 (2020)
19. M. Malita, Membrane computing in Prolog, in *Pre-Proceedings of the Workshop on Multiset Processing, Curtea de Arges, Romania, TR 140, CDMTCS*, ed. by C.S. Calude, M.J. Dinneen, Gh. Păun (University of Auckland, Auckland, 2000), pp. 159–175
20. M.A. Martínez-del-Amor, M. García-Quismondo, L.F. Macías-Ramos, L. Valencia-Cabrera, A. Riscos-Núñez, M.J. Pérez-Jiménez, Simulating P systems on GPU devices: a survey. Fundam. Inform. **136**(3), 269–284 (2015). https://doi.org/10.3233/FI-2015-1157
21. M.A. Martínez-del-Amor, L.F. Macías-Ramos, L. Valencia-Cabrera, M.J. Pérez-Jiménez, Parallel simulation of population dynamics P systems: updates and roadmap. Nat. Comput. **15**(4), 565–573 (2016). https://doi.org/10.1007/s11047-016-9566-1
22. V.T.T. Nguyen, *An Implementation of the Parallelism, Distribution and Nondeterminism of Membrane Computing Models on Reconfigurable Hardware* (University of South Australia, Australia, 2010)
23. V.T.T. Nguyen, D. Kearney, G. Gioiosa, An algorithm for non-deterministic object distribution in P systems and its implementation in hardware, in *Membrane Computing (WMC 2008)*, ed. by D.W. Corne, P. Frisco, Gh. Păun, G. Rozenberg, A. Salomaa. Lecture Notes in Computer Science, vol.5391 (2008), pp. 325–354. https://doi.org/10.1007/978-3-540-95885-7_24
24. V.T.T. Nguyen, D. Kearney, G. Gioiosa, An implementation of membrane computing using reconfigurable hardware. Comput. Inf. **27**(3+), 551–569 (2008)
25. V.T.T. Nguyen, D. Kearney, G. Gioiosa, A region-oriented hardware implementation for Membrane Computing applications, in *Membrane Computing (WMC 2009)*, ed. by Gh. Păun, M.J. Pérez-Jiménez, A. Riscos, G. Rozenberg, A. Salomaa. Lecture Notes in Computer Science, vol. 5957 (2010), pp. 385–409. https://doi.org/10.1007/978-3-642-11467-0_27
26. Gh. Păun, Computing with membranes, in *Technical Report* (Turku Centre for Computer Science, Turku, 1998)
27. Gh. Păun, Computing with membranes. J. Comput. Syst. Sci. **61**(1), 108–143 (2000). https://doi.org/10.1006/jcss.1999.1693
28. Gh. Păun, *Membrane Computing, An Introduction* (Springer, Berlin, 2002)
29. Gh. Păun, G. Rozenberg, A. Salomaa, *The Oxford Handbook of Membrane Computing* (Oxford University, Oxford, 2010)
30. B. Petreska, C. Teuscher, A reconfigurable hardware membrane system, in *Membrane Computing (WMC 2003)*, ed. by C. Martín-Vide, G. Mauri, Gh. Păun, G. Rozenberg, A. Salomaa. Lecture Notes in Computer Science, vol. 2933 (2003), pp. 269–285. https://doi.org/10.1007/978-3-540-24619-0_20
31. J. Quirós, *Implementación Sobre Hardware Reconfigurable de una Arquitectura no Determinista, Paralela y Distribuida de Alto Rendimiento, Basada en Modelos de Computación con Membranas*, Ph.D. thesis (Universidad de Sevilla, Sevilla, 2015, in Spanish). http://hdl.handle.net/11441/39088
32. J. Quirós, S. Verlan, J. Viejo, A. Millán, M.J. Bellido, Fast hardware implementations of static P systems. Comput. Inf. **35**(3), 687–718 (2016)
33. S. Raghavan, K. Chandrasekaran, Tools and simulators for membrane computing: a literature review, in *Bio-inspired Computing–Theories and Applications (BIC-TA 2016)*, ed. by M. Gong, L. Pan, T. Song, G. Zhang. Communications in Computer and Information Science, vol. 681 (Springer, Singapore, 2016), pp. 249–277. https://doi.org/10.1007/978-981-10-3611-8_23
34. S. Sedwards, T. Mazza, Cyto-Sim: a formal language model and stochastic simulator of membrane-enclosed biochemical processes. Bioinformatics **23**(20), 2800–2802 (2007). https://doi.org/10.1093/bioinformatics/btm416
35. Z. Shang, *Hardware Implementation of Cell-inspired Computational Models*. Ph.D. Thesis (University Paris-Est Créteil Val de Marne, Paris, 2020)
36. Z. Shang, S. Verlan, G. Zhang, Hardware implementation of numerical P systems, in *Proceedings of the 20th International Conference on Membrane Computing, CMC20, August 5–8, 2019*,ed. by Gh. Păun (Curtea de Arges, Romania, 2019), pp. 463–474

37. Z. Shang, S. Verlan, G. Zhang, H. Rong, FPGA implementation of numerical P systems. Int. J. Unconv. Comput. **16**(2–3), 279–302 (2021)
38. Y. Suzuki, H. Tanaka. On a LISP implementation of a class of P systems. Rom. J. Inf. Sci. Technol. **3**(2), 173–186 (2000)
39. A. Syropoulos, E.G. Mamatas, P.C. Allilomes, K.T. Sotiriades, A distributed simulation of transition P systems, in *Membrane Computing (WMC 2003)*, ed. by C. Martín-Vide, G. Mauri, Gh. Păun, G. Rozenberg, A. Salomaa. Lecture Notes in Computer Science, vol. 2933 (2004), pp. 357–368. https://doi.org/10.1007/978-3-540-24619-0_25
40. L. Valencia-Cabrera, D. Orellana-Martín, M.A. Martínez-del-Amor, M.J. Pérez-Jiménez, An interactive timeline of simulators in Membrane Computing. J. Membr. Comput. **1**, 209–222 (2019). https://doi.org/10.1007/s41965-019-00016-z
41. L. Valencia-Cabrera, I. Pérez-Hurtado, M.A. Martínez-del-Amor, Simulation challenges in membrane computing. J. Membr. Comput. **2**(4), 392–402 (2020). https://doi.org/10.1007/s41965-020-00056-w
42. S. Verlan, J. Quirós, Fast hardware implementations of P systems, in *Membrane Computing (CMC 2012)*, ed. by E. Csuhaj-Varjú, M. Gheorghe, G. Rozenberg, A. Salomaa, G. Vaszil. Lecture Notes in Computer Science, vol. 7762 (2012), pp. 404–423. https://doi.org/10.1007/978-3-642-36751-9_27
43. G. Zhang, M.J. Pérez-Jiménez, M. Gheorghe, *Real-life Applications with Membrane Computing* (Springer, Berlin, 2017)
44. G. Zhang, Z. Shang, S. Verlan, M.A. Martínez-del-Amor, C. Yuan, L. Valencia-Cabrera, M.J. Pérez-Jiménez, An overview of hardware implementation of Membrane Computing models. ACM Comput. Surv. **53**(4), Article No. 90, 1–38 (2020). https://doi.org/10.1145/3402456