# A Simulation Workflow for Membrane Computing: From MeCoSim to PMCGPU Through P-Lingua

Luis Valencia-Cabrera, Miguel Á. Martínez-del-Amor, and Ignacio Pérez-Hurtado

Research Group on Natural Computing, Department of Computer Science and Artificial Intelligence,
Universidad de Sevilla, Avda. Reina Mercedes s/n, 41012 Seville, Spain
{lvalencia,mdelamor,perezh}@us.es

**Abstract.** P system simulators are of high importance in Membrane Computing, since they provide tools to assist on model validation and verification. Keeping a balance between generality and flexibility, on the one side, and efficiency, on the other hand, is always challenging, but it is worth the effort. Besides, in order to prove the feasibility of P system models as practical tools for solving problems and aid in decision making, it is essential to provide functional mechanisms to have all the elements required at disposal of the potential users smoothly integrated in a robust workflow. The aim of this paper is to describe the main components and connections within the approach followed in this pipeline.

## 1 Introduction

Since the early days of Membrane Computing two decades ago [17], the *in-depth* study of different types and variants of its computational devices, the *so-called* P systems, in terms of their computational power and complexity, has come along with the interest in the development of simulation tools to explore their properties and explode some of their most promising features, as extensively described in [21].

While the full potential and practical ability of P systems could only be exploited so far through the real implementation of some variants of these machine-oriented computing devices, the development of software simulators has always being considered worth exploring [4]. The main uses of these tools have ranged from pedagogical to research assistants to help studying properties of the computing devices and the solutions they aim to solve at both theoretical and practical levels, playing a crucial role in the design and verification of such solutions. Additionally, they have been successfully applied to aid in the decision-making process to manage real-world situations in ecological and industrial systems, providing useful predictive insights when virtually experimenting with potential scenarios making use of simulators for the corresponding models based on P systems.

Within this overall scenario, the approach followed from our research group has been the coverage of a wide range of membrane computing devices. Not only the range of applications where we have applied simulators has been as broad as we could, but also the tools developed tried from the beginning to be as general, flexible, reusable and extensible as we could achieve. It was not the short-term publication that leaded our research and development efforts, but it was the intention of providing something the community could take advantage of, both for theoretical studies and practical applications. In this sense, the main lines explored have been directed towards three main ideas:

(1) **generality**, with the design of a standard language to specify P systems (`P-Lingua`), along with parsing, debugging and simulation tools within `P-Lingua` framework [5,19];
(2) **practical use**, through the development of more user-oriented tools to manage the models for both designers with a higher-level interface and additional exploration tools, and end users using custom applications based on `MeCoSim` [18,20] as black boxes; and
(3) **efficiency**, looking for appropriate architectures and designs in the development of software tools, that could (at least partially) benefit from the inherent parallelism in the essence of P systems through high performance platforms, as pursued by `PMCGPU` project [10,24].

The work presented here intends to put together all the achievements accomplished by these three main lines introduced above, and clarify the potential users about the main features and guidelines to use effectively the tools developed, hence providing new results presenting a good balance in terms of compromise between generality and efficiency, always bearing in mind practicality. In order to achieve these goals, a simulation workflow is proposed, starting from the design of a model and a customized application and finishing with the simulation in high performance platforms, as we will explain later.

The rest of the paper is distributed as follows. After this section where the general intention has been declared, Sect. 2 introduces the main features of the essential elements involved in the process just outlined. Then, in Sect. 3 the simulation workflow will be described in detail, clarifying the crucial points of the proposal and the connection among all the different ingredients taking part on this recipe. Finally, Sect. 4 summarises the most relevant achievements of this work, with some conclusions shedding some light upon the benefits, use and possible applications where this approach may help in future works.

## 2 Simulation Software for P Systems

This section introduces the roles played by the main elements of the workflow proposed in this work, highlighting the most important features of the tools provided by each project and some practical aspects of their use.

## 2.1 The Visual Environment MeCoSim

As mentioned in the introduction of this paper, one of the main aspects of the simulation in membrane computing has always been the practical application of the designs for relevant theoretical and real-life problems. This involves, in addition to solid foundations (given by the study of the computing devices in terms of computational power and complexity), the development of simulation tools aiming not only to allow technical people to develop solutions based on P systems, but also making their life easier when dealing with them, and also enabling end users to take advantage of those solutions as black boxes. This way, P-system based models for solving real-world problems could play a relevant role for this end users, assisting them in their decisions, permitting the experimentation with their scenarios of interest without further knowledge about the internal details. Regarding P systems designers, they should be able to have at their disposal some tools to handle their designs in a friendly way, with facilities to: check the correctness of their models, both from a lexical-syntactic and semantic view; populate different instances of the P systems designed; visualize properly the structure and contents of the elements created during the initialization and the step-by-step run of the simulations, and in general provide useful options helping in the tough tasks of design and verification of solutions based on P systems.

From the very beginning in 2009–2010 [18], `MeCoSim` was designed with the two-fold intention expressed above, to help P systems designers with the modelling and verification, and also allowing end users to have solutions in membrane computing. For the first goal, this software was always supported by the parsing, debugging and simulation engine given by `P-Lingua`, while the latter objective required further development to provide some mechanism where P systems designers could easily prepare end-user applications by a relatively simple configuration. Thus, `MeCoSim` was initially designed to enable the user defined customized interfaces, with inputs, outputs, charts, etc., adapted to each model or solution given by a family of P systems, expressed in `MeCoSim` language. This enables the users to enter their data for different initial conditions, instantiating the desired P systems within the family.

Apart from the initial goals, this software environment had from its conception two essential objectives: the flexibility required for the definition of any kind of custom applications and variety of P systems accepted, and the extensibility to increase its initial functionalities through the development using its plugins architecture and the integration with external packages. Thus, the seminal version of the software was enriched by the integration with different tools for the property extraction and verification of P systems [6,9] and the development of plugins for graph-based problems, definition of propositional formulas, extensions of the language accepted by the generation of parameters, connection with external simulators based on `Spin` [7] and several other new features for designers and end users. Extensive documentation, case studies, explicative videos, etc. about everything related with MeCoSim can be found at [22].

## 2.2 P-Lingua, and the pLinguaCore Library

In the heart of our proposal is `P-Lingua framework` [5,23], the undeniable cornerstone of the approach. It was conceived from the beginning as an ambitious open source project within the area of Membrane computing, aiming to provide general-purpose tools for its research community, rather than focusing on specific short-term results.

Among its main goals, this project included a standard language, `P-Lingua`, for the specification/definition of P systems, for a variety if types and variants of such systems, through a unifying set of operators. As extensively described in [19], this language is modular, favouring structured programming, and provides a syntax very close to the format employed in the description of P systems we can find in research. In addition to the language and the tools developed to process its plain text files (with `.pli` extension), a whole framework was surrounding this core with debugging tools, some sp-called input and output parsers, and a complete set of simulators for the variants supported by the different versions of `pLinguaCore` library, the main software product including the crucial part of the framework. Some independent programs allow the compilation and run of models and solutions based on the library, starting with some useful command-line tools and finishing with MeCoSim.

Furthermore, inside the framework there is a parser that performs lexical and syntactical analysis to text files describing models in P-Lingua [5]. This is used to initialize internal structures that can serve later for the simulator in pLinguaCore or to generate an output file, in a different format, describing the same model. So far, XML exportation is supported for the majority of models. For the ones supported by `PMCGPU`, an specific ad-hoc binary format is generated, which aims at being a communication language between P-Lingua and the parallel standalone simulators. Moreover, a straightforward semantics processing is carried out after the syntactical analysis, in the sense that the P system features correspond to the model specified.

*pLinguaCore* is a framework that makes use of software design patterns to make the development of new simulators easier [5,19]. Modularity and flexibility are the key aspects of it. Written in Java, and together with the parser, this framework provides a complete generic environment for P system simulations, from their specification, parsing, simulation and output. For each P system model, it is possible to implement different engines. We should note that the objective of pLinguaCore is not efficiency, so it might happen that the simulation of some models should take a considerable time.

## 2.3 PMCGPU for Accelerated Simulations

Parallel simulation of P systems is becoming a hot topic within Membrane Computing. The need of efficient simulators, along with the interest of implementing bio-inspired parallelism using High Performance Computing, has increased the interest on it. Latest attempts to parallelize simulations have been based on GPU computing [10,11]. The multiprocessors of graphics cards provide a parallel architecture that is being harnessed as enabling technologies for many applications.

Since the introduction of CUDA in 2007 [8], parallel simulators on GPUs have been developed. The main project that covers these developments is `PMCGPU` (Parallel simulators for Membrane Computing on the GPU) [10,24]. Within this open-source project, one can find simulators for P systems with active membranes (codenamed PCUDA) [2,11] and for Population Dynamics P systems (ABCDGPU) [11–13,16]. There are also two specific simulators for two solutions to `SAT` problem, one based on P systems with active membranes (PCUDASAT) [3,11] and other on tissue–like P systems with cell division (TSPCUDASAT) [11,14]. Finally, simulators for spiking neural P systems have been recently attached to the project (cuSNP) as external developments.

Concerning PCUDA and ABCDGPU, both simulators receive as input a P system description, and return as output either a description in plain text of each transition step, or a CSV (comma–separated–values) with the information of the whole simulation (this last mode is available only in ABCDGPU [12]). The description of the P system to be simulated in both cases is based on an specific binary file. Further information about the structure of the input binary files is available in [10]. In order to ease the workflow, the binary files can be generated using pLinguaCore from a P-Lingua file, as described in the simulation workflow in Sect. 3.

In the following sub-sections, an overview of PCUDA and ABCDGPU is provided, together with general instructions for their installation, and also a short introduction to core concepts of GPU computing.

**GPU Computing.** Graphics Processing Units (GPUs) have evolve in the past 15 years in such a way that, today, they serve as a massively parallel co-processor for heterogeneous computing [8]. In 2007, NVIDIA introduced a new programming model that abstracts the underlying architecture of GPUs, so programmers only need to think in threads and in a memory hierarchy. Parallel programming on a GPU is like a game in which the amount of threads has to be balanced (the more the better, but doing effective work) and the access to memory has to be done by maximizing the usage of registers and cache, plus implementing coalesced access to data (contiguous threads loading consecutive positions in memory).

Threads are executed hierarchically, being arranged into synchronizing and cooperating blocks. Thread blocks also include a small but fast memory, called shared memory, that is used to efficiently store frequently used data. Current GPUs provide thousands of cores grouped in multiprocessors, and a dozen of Gigabytes of memory. The CUDA scheduler automatically assign blocks to multiprocessors, and groups of 32 threads, called warp, are launched simultaneously to the cores. All threads execute the same code, which is a function called kernel. The CPU is the one in charge of launching kernels, and of sending and retrieving data from the GPU [8].

Best design practices are to launch as much threads as possible, from 128 to 512 threads per block, making use of a reasonable amount of shared memory, let contiguous threads access contiguous portions of data, and avoid thread divergence (i.e. threads within a warp executing different instructions).

**PCUDA: Parallel Simulation of P Systems with Active Membranes.** The first simulator for P systems on CUDA was PCUDA [2]. In this work, the two-level parallel nature of P systems is mapped to the two-level parallelism on GPUs: membranes are assigned to thread blocks, and rules are processed by threads. Since the simulated model has no cooperation (only one object in the LHS), threads are effectively processing objects. In order to ease this work assignation, several assumptions from the input P system are taken:

(1) models are confluent, so that all computation halt and lead to the same result;
(2) only two levels in the membrane hierarchy is supported; that is, only a skin membrane and elementary membranes; and
(3) the amount of objects defined in the alphabet must be multiple of a number below 512.

The two last conditions are more technical issues, but that we plan to solve in future releases. The first condition is an important restriction for the design of the simulator: if the P system is confluent, then the computation path does not matter, and the simulator can choose to take the "cheapest" one (in terms of amount of membranes generated and evolution rules applied).

When simulating this P system, the major performance attribute is related to the object density [11]: the GPU simulator runs faster as long as more different objects appear in the regions of the P systems. If the amount of different objects is small, then the majority of threads will be idle. This can be seen from the two benchmarks executed with the simulator in [10]: a toy example designed to stress the simulator (up to 7x of speedup with a Tesla C1060), and a linear solution to `SAT` (up to 1.67x of acceleration).

The simulator iteratively carries out the transitions of the model until a stop configuration is reached. Moreover, it is optional to retrieve just the last configuration or the whole computation reproduced by the simulator. The input of the simulator is given by a specific binary format which is coupled to the way the internal data structures of the simulator are created. In this way, reading the file and creating the membrane and rule representations are efficiently carried out. The latest version (to date) of the binary format is available in [10].

There are two ways of generating a binary file for this simulator: manually creating a binary file (not recommended), or using pLinguaCore to create a binary file from a P-Lingua file (recommended). A version of pLinguaCore is redistributed along with PCUDA, including some examples and guiding files. The specific command to create a binary file can be seen in Sect. 3.2. Finally, 3 versions of this simulator is included in PCUDA: a slow but general sequential, slow but restricted fast and a parallel GPU but restricted simulator. The term restricted means that the last two assumptions of the three aforementioned are taken.

The source code is available in [24], and it can be installed by following the next instructions:

1. Install a CUDA Toolkit version 3.X - 4.X in the computer, and install locally the CUDA SDK. Set the `PATH` and `LD_LIBRARY_PATH` environment variables accordingly.
2. Install the counterslib package: extract the file counterslib.tar.gz inside the *common* folder of the CUDA SDK, go inside the folder and type `./compile.sh`..
3. Unzip the file pcuda-1.0.tar.gz into the CUDA SDK folder *C/src*. Go inside the folder and type `make`. You should see the executable file inside the folder, or in *../../bin/linux/release*.

**ABCDGPU: Parallel Simulation of Population Dynamics P Systems.** Simulating PDP systems [1] in parallel is becoming critical, given that some models require to handle such amount of elements that sequential simulators do not give a result in an acceptable time. Although workarounds have been done to tune models and escape from performance pitfalls, parallel simulators might enable the simulation of large models with accurate algorithms such as DCBA [15]. In [13], a parallel implementation of DCBA was carried out for multicore processors, and in [16], for GPUs. Best performance was achieved with the latter, and yet using old GPU architectures. The simulator was experimentally validated with a model of the Bearded Vulture at the Catalan Pyrinean in [12].

The GPU design for ABCDGPU is parallelized by simulations (assuming that the user wants to make an statistical study from the probabilistic model) and environments. Given that rules have cooperation (more than one object appearing in the left-hand side), even between two membranes (the active membrane and its parent), the parallelization of membrane processing is voided. In turn, threads iterate over the rule blocks in tiles (groups of $N = 256$), and in each total iteration the threads execute a step of the DCBA over them. In DCBA, a distribution table is employed, but given that this table can be sparse, the simulator uses a virtual table which is based only on the rule information and two extra arrays (for row additions and column minimum calculation) [13]. Phase 1 to 3 of DCBA follows this distribution, while phase 4 parallelize rules instead of rule blocks along threads [10].

The performance of this simulator also depends on the object density, given that the membrane representation assumes that all different objects might appear at certain point [11]. For random number generation, a new library called CURNG_BINOMIAL was implemented [16]. Although this library creates unfortunately a high thread divergence, the performance is enough for most models. When the ratio of rule blocks competition is high (blocks having different but overlapping left hand sides), phase 2 becomes a bottleneck. In any case, a sequential, a multicore and a GPU simulator are available within ABCDGPU, so researchers can choose a version depending on the resources available. Reported speedups are up to 7x with stressing experiments [16], and up to 5x with a real but small ecosystem [12].

Latest version of ABCDGPU supports the generation of a CSV formatted file with the whole information of the simulator [12], so it might be possible to load it in other tools for its analysis (e.g. Excel/Calc, R, SQL, `MeCoSim`, etc.). As in PCUDA, in order to speedup the initialization of the simulator, a binary format is employed for the input [10]. It is coupled to the way the internal data structures are created, so there is no need of buffering information or doing extra loops, because the required information for the next item to be allocated is always available in the traversing of the file. In order to avoid the generation of binary files manually, pLinguaCore was extended with a binary generator for PDP systems. Hence, a P-Lingua file can be used as an input. A redistributable version of pLinguaCore with this binary module comes along with ABCDGPU, and the corresponding command line is described in Sect. 3.2.

The source code can be downloaded from [24], and the installation can be made by following the next instructions:

1. Install a CUDA Toolkit version 5.X or later, and install locally the CUDA SDK (to date, up to version 9.1 has been tested). Set the `PATH` and `LD_LIBRARY_PATH` environment variables accordingly.
2. Install the GNU Scientific Library (GSL) and Electric Fence.
3. Inside the folder of CUDA SDK samples, create a new folder named *8_pmcgpu*.
4. Extract the contents of abcd-gpu-1.0b.tar.gz and counterslib.tar.gz files into a new folder *8_pmcgpu/ abdcd-gpu*.
5. Go to folder *abcd-gpu*, and type `make`. You should see the executable file inside the folder.

## 3 A Complete Workflow

This section describes in detail the main components of the pipeline proposed in this work, along with the connections between them.

### 3.1 Model Personalization with MeCoSim

As part of the **workflow** described in this paper, given a certain problem we aim to solve, the role played by `MeCoSim` is the end-point where the end user will interact with the P systems designed and the simulation tools behind the scene. On the one hand, it will provide the mechanism to *configure a custom application* to handle the solution to the problem under study, where the designer will be able to debug its model and, once it is validated, the end users will run their simulations.

Thus, for a given design of a family of P system solving a certain problem, P systems designer provide a configuration file defining the custom app (in .xls or .json format), along with a P-Lingua file with the design based on the type and variant of membrane computing device chosen. This elements will be loaded in

MeCoSim, so that the end user can open the app and start running experiments. In the regular use of the application by these end users, whenever they introduce a new scenario through visual input tables in their custom interface, the data introduced is processed to generate parameters involved in the instantiation of the specific P system within the family of P systems specified in MeCoSim.

Given the specific P system instantiated, depending on the simulator selected, the system will run inside MeCoSim framework or will call some external simulator configured to run with MeCoSim. All simulators implemented within MeCoSim framework provide a step method that is called by MeCoSim consecutively until a halting condition is reached or until the given number of simulations, cycles and steps have finished. However, we cannot guarantee that every external simulator will provide the same mechanism or if this repetitive call will compromise the efficiency of the simulation. Therefore, the connection with the external simulators is based on the independent run of those simulation engines and the setting of protocols to generate the results of the simulation in some agreed format that can be later loaded in MeCoSim, thus populating the output tables and charts configured in the custom applications based on MeCoSim, remaining independent on the simulation engines ultimately used to run the simulations and their internal details, and focused instead on the user view of the information returned by the solution designed.

When the external simulators are not completely integrated (in terms of automation) with the framework provided by P-Lingua and MeCoSim, which is currently the case in many projects, we can still take advantage of the different capabilities of MecoSim to define the custom end-user application and run the models. Thus, once the P-Lingua specification for our problem is loaded in MeCoSim and the data about the scenario under interest is introduced by the end user in the visual input tables, the syntactic and semantic correctness of the file and input data is checked the specific instance of the problem (with the corresponding P system of the family behind) is created. Then, inside MeCoSim folder (mecosim-rgnc, in our home directory), a new file (modelname-parameters.pli) is automatically generated (in userfiles subfolder). Then, the specific P system can be defined as the pair consisting of the original modelname.pli representing the whole family providing the solution, plus the modelname-parameters.pli providing the accompanying parameters for the specific scenario introduced by the user. Th junction of both files will be ready to be used in th following steps of our workflow.

### 3.2 Simulation with PMCGPU Through P-Lingua

As discussed in Sect. 2.3, PMCGPU incorporates simulators for two P system models: active membranes and Population Dynamics. The other simulators are adhoc for specific solutions to SAT. Therefore, the workflow has two paths, depending on the P system model to be simulated.

**P Systems with Active Membranes.** In order to simulate a P system with active membranes with PCUDA, first, a P-Lingua file describing the model is required. In the previous step of this workflow, it is explained how a P-Lingua file can be generated with `MeCoSim`, which includes all required parameters. This generated file has to be copied to the computer where PCUDA has been installed, inside the folder *pcuda/plingua*. The binary file can be created out of the P-Lingua one by executing the following instruction (please replace "inputFile" by the name of the corresponding P-Lingua file):

```
java -jar pparser.jar inputFile.pli inputFile.bin
```

Once a binary file has been generated without errors (in *pcuda/plingua/? inputFile.bin*, if the last instruction was executed literally), it is time to take the next step and use the simulator. In the current version of PCUDA, we have to provide some help to the simulator and define:

– #membranes: Number of membranes that the model will generate (an upper bound)
– #objects: Number of objects defined in the alphabet
– #threadPerBlock: Number of threads per block, defined as $T$ such that $T*n = \#objects$ and $0 < T \leq 512$.

Having at hand all of this information, we can then launch the simulator by one of the following instructions (depending on the version to run):

– The sequential simulation: `./pcuda -s -i inputFile.bin`
– The fast sequential simulation: `./pcuda -f -i inputFile.bin -m #membranes -o #objects -b #threadPerBlock -t 1`
– The parallel simulation on the GPU: `./pcuda -p -i file.bin -m #membranes -o #objects -b #threadPerBlock -t 1`

If the computer has more than one GPU, it is possible to choose which one will run the simulator, by setting up the environment variable by typing `export DEFAULT_DEVICE=x`, where $x$ is the id of the GPU to use for the simulation. This id can be consulted by using the tool *nvidia-smi* or the SDK example *deviceQuery*). A limit to the number of transition steps can be imposed using the parameter "-l". In order to store the whole output of the simulator, just use the standard output forwarding as > *outputFile.txt*. Finally, one can choose to show all transition steps or just the final one with the verbosity parameter "-v 2" or "-v 1", respectively. The format uses a similar syntax than the output given by pLinguaCore. For more information, show the help dialogue by typing `./pcuda -h`.

**Population Dynamics P Systems.** Once a complete P-Lingua file (including all parameter definitions) has been generated with `MeCoSim`, we have all at hand to carry out the simulation with ABCDGPU. Next step is to copy the desired P-Lingua file into the computer where the GPU simulator has been installed, and place it in the folder *abcd-gpu/plingua*. In order to generate the binary file with pLinguaCore, the next command has to be executed (replace "inputFile" with the corresponding file name):

`java -jar pLinguaCore.jar plingua inputFile.pli -bin inputFile.bin`

If the binary has been generated without errors, the following parameters have to be chosen:

- #simulations: Number of simulations to run (around 25–100).
- #steps: Number of transitions steps to run.
- #cycles: Number of transitions that conform a cycle. The simulator will output information only after every cycle.
- #cores: Only if the multicore simulator is used, select the amount of cores to use in the system.

Once the parameters have been set, it is time to execute the simulator. Use one of the following instructions to run one version:

- The sequential simulator: `./abcdgpu -f plingua/inputFile.bin -s #simulations -t #steps -v 1 -c #cycles -O 0`
- The multicore simulator: `export OMP_NUM_THREADS=#cores; ./abcdgpu -f plingua/inputFile.bin -s #simulations -t #steps -v 1 -c #cycles -O 0`
- The GPU simulator: `./abcdgpu -f plingua/inputFile.bin -I 1 -s #simulations -t #steps -v 1 -c #cycles -O 0`

In all cases, a CSV file is generated with the information at every end of a cycle (set #cycle = 1 to output at every transition step). The name of the output file is equal to the input file but with .csv extension. This CSV file can be processed by any platform supporting this format: R, Python, C, Excel/Calc, etc. In the near future, this output file can be reincorporated into `MeCoSim`, so similar analysis can be performed than when using only pLinguaCore within the simulation framework. Finally, the CSV file contains the following columns: *SIMULATION, STEP, ENVIRONMENT, MEMBRANE, OBJECT, MULTIPLICITY*. When deleting the parameter "-O 0", the output will be a plain text file describing every transition, as made in PCUDA. The format uses a similar syntax than the output given by pLinguaCore.

## 4 Conclusions and Perspectives

Simulation tools are indispensable in Membrane Computing. In this concern, software developments within the Research Group on Natural Computing at the University of Seville have led to three main projects with different aims: `MeCoSim` (for applications of Membrane Computing), `P-Lingua` (for flexible description and simulation of P systems), and `PMCGPU` (for accelerating P system simulations with GPUs and parallel platforms).

In this paper, we have shown a workflow to interconnect all of them in order to perform fast simulations of P systems (for either with active membranes and PDP systems). It goes from the top level, at the applications and end users without P system expertise, to the bottom level, executing simulations on parallel hardware. P-Lingua acts as a wrapper to communicate both sides. Future work involves further developments and implementations of models, more applications and an automatic integration of all the tools, so that the whole pipeline can run efficiently and transparently to end users.

**Acknowledgments.** The authors are very grateful to Mario J. Pérez-Jiménez for his unconditional support, unlimited generosity, patience and enthusiasm, and particularly for his skilful advising and guiding as their "scientific father".

The authors also acknowledge the support from the research project TIN2017-89842-P, cofinanced by "Ministerio de Economía, Industria y Competitividad" (MINECO) of Spain, through the "Agencia Estatal de Investigación" (AEI), and by "Fondo Europeo de Desarrollo Regional" (FEDER) of the European Union.

## References

1. Cardona, M., et al.: A computational modeling for real ecosystems based on P systems. Nat. Comput. **10**(1), 39–53 (2011)
2. Cecilia, J.M., García, J.M., Guerrero, G.D., Martínez-del-Amor, M.A., Pérez-Hurtado, I., Pérez-Jiménez, M.J.: Simulation of P systems with active membranes on CUDA. Brief. Bioinform. **11**(3), 313–322 (2010)
3. Cecilia, J.M., García, J.M., Guerrero, G.D., Martínez-del-Amor, M.A., Pérez-Hurtado, I., Pérez-Jiménez, M.J.: Simulating a P system based efficient solution to SAT by using GPUs. J. Log. Algebr. Program. **79**(6), 317–325 (2010)
4. Díaz-Pernil, D., Graciani, C., Gutiérrez-Naranjo, M.A., Pérez-Hurtado, I., Pérez-Jiménez, M.J.: Software for P systems. In: Păun, Gh., Rozenberg, G., Salomaa, A. (eds.) The Oxford Handbook of Membrane Computing. Oxford University Press, pp. 437–454 (2009)
5. García-Quismondo, M., Gutiérrez-Escudero, R., Pérez-Hurtado, I., Pérez-Jiménez, M.J., Riscos-Núñez, A.: An overview of P-Lingua 2.0. In: Păun, G., Pérez-Jiménez, M.J., Riscos-Núñez, A., Rozenberg, G., Salomaa, A. (eds.) WMC 2009. LNCS, vol. 5957, pp. 264–288. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-11467-0_20
6. Gheorghe, M.: 3-COL problem modelling using simple Kernel P systems. Int. J. Comput. Math. **90**(4), 816–830 (2013)
7. Ipate, F., et al.: Kernel P systems: applications and implementations. Adv. Intell. Syst. Comput. **212**, 1081–1089 (2013)
8. Kirk, D.B., Wen-Mei, W.H.: Programming Massively Parallel Processors: A Hands on Approach. Morgan Kauffman (2010)
9. Lefticaru, R., et al.: Towards an integrated approach for model simulation, property extraction and verification P systems. In Martínez, M.A., Paun, Gh. Pérez, I., Romero, F.J. (eds.) Proceedings of the Tenth Brainstorming Week on Membrane Computing, vol. 1, pp. 291–318 (2012)
10. Martínez-del-Amor, M.A.: Accelerating membrane systems simulators using high performance computing with GPU, Ph.D. thesis, University of Seville (2013)
11. Martínez-del-Amor, M.A., García-Quismondo, M., Macías-Ramos, L.F., Valencia-Cabrera, L., Riscos-Núñez, A., Pérez-Jiménez, M.J.: Simulating P systems on GPU devices: a survey. Fundam. Inform. **136**(3), 269–284 (2015)
12. Martínez-del-Amor, M.A., Macías-Ramos, L.F., Valencia-Cabrera, L., Pérez-Jiménez, M.J.: Parallel simulation of Population Dynamics P systems: updates and roadmap. Nat. Comput. **15**(4), 565–573 (2016)

13. Martínez-del-Amor, M.A., Karlin, I., Jensen, R.E., Pérez-Jiménez, M.J., Elster, A.C.: Parallel simulation of probabilistic P systems on multicore platforms. In: García, M., Macías, L.F., Păun, Gh., Valencia, L. (eds.) Proceedings of the Tenth Brainstorming Week on Membrane Computing (BWMC 2012), vol. 2, pp. 17–26 (2012)
14. Martínez-del-Amor, M.A., Pérez-Carrasco, J., Pérez-Jiménez, M.J.: Characterizing the parallel simulation of P systems on the GPU. Int. J. Unconv. Comput. **9**(5–6), 405–424 (2013)
15. Martínez-del-Amor, M.A., et al.: DCBA: simulating population dynamics P systems with proportional object distribution. In: Csuhaj-Varjú, E., Gheorghe, M., Rozenberg, G., Salomaa, A., Vaszil, G. (eds.) CMC 2012. LNCS, vol. 7762, pp. 257–276. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36751-9_18
16. Martínez-del-Amor, M.A., Pérez-Hurtado, I., Gastalver-Rubio, A., Elster, A.C., Pérez-Jiménez, M.J.: Population dynamics P systems on CUDA. In: Gilbert, D., Heiner, M. (eds.) CMSB 2012. LNCS, pp. 247–266. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33636-2_15
17. Păun, G.: Computing with membranes. J. Comput. Syst. Sci. **61**(1), 108–143 (2000). And Turku Center for Computer Science-TUCS Report No 208
18. Pérez-Hurtado, I., Valencia-Cabrera, L., Pérez-Jiménez, M.J., Colomer, M.A., Riscos-Núñez, A.: MeCoSim: a general purpose software tool for simulating biological phenomena by means of P systems. In: Li, K., Tang, Z., Li, R., Nagar, A.K., Thamburaj, R. (eds.) IEEE Fifth International Conference on Bio-inspired Computing: Theories and Applications (BIC-TA 2010), vol. 1, pp. 637–643 (2010)
19. Pérez-Hurtado, I.: Desarrollo y aplicaciones de un entorno de programación para Computación Celular: P-Lingua. Ph.D. thesis. University of Seville (2010)
20. Valencia-Cabrera, L.: An environment for virtual experimentation with computational models based on P systems. Ph.D. thesis. University of Seville (2015)
21. Valencia-Cabrera, L., Orellana-Martín, D., Martínez-del-Amor, M.A., Pérez-Jiménez, M.J.: From super-cells to robotic swarms: two decades of evolution in the simulation of P systems. Bull. Int. Membr. Comput. Soc. Number **4**, 65–87 (2017)
22. MeCoSim website. http://www.p-lingua.org/mecosim
23. The P-Lingua website. http://www.p-lingua.org
24. The PMCGPU project website. http://sourceforge.net/p/pmcgpu