Parallel simulation of Population Dynamics P systems: updates and roadmap

Miguel A. Martínez-del-Amor,[1] Luis F. Macías-Ramos,[1] Luis Valencia-Cabrera,[1] Mario J. Pérez-Jiménez[1]

Abstract Population Dynamics P systems are a type of multienvironment P systems that serve as a formal modeling framework for real ecosystems. The accurate simulation of these probabilistic models, e.g. with *Direct distribution based on Consistent Blocks Algorithm*, entails large run times. Hence, parallel platforms such as GPUs have been employed to speedup the simulation. In 2012, the first GPU simulator of PDP systems was presented. However, it was able to run only randomly generated PDP systems. In this paper, we present current updates made on this simulator, involving an input modu le for binary files and an output module for CSV files. Finally, the simulator has been experimentally validated with a real ecosystem model, and its performance has been tested with two high-end GPUs: Tesla C1060 and K40.

Keywords Membrane computing · Ecological modelling · PDP systems · Parallel simulation · GPU computing · CUDA

✉ Miguel A. Martínez-del-Amor
mdelamor@us.es

Luis F. Macías-Ramos
lfmaciasr@us.es

Luis Valencia-Cabrera
lvalencia@us.es

Mario J. Pérez-Jiménez
marper@us.es

[1] Research Group on Natural Computing, Universidad de Sevilla, Avda. Reina Mercedes s/n, 41012 Seville, Spain

1 Introduction

P systems (Păun 2000; Păun et al. 2010) have become good candidates for computational modeling thanks to the compartmental and discrete features, both in Systems Biology (Pérez-Jiménez and Romero-Campero 2006; Romero-Campero and Pérez-Jiménez 2008) and Population Dynamics (Colomer-Cugat et al. 2014). In this sense, it is worth to mention the achieved success in real ecosystem modeling through probabilistic P systems, such as the case of Bearded Vulture in the Catalan Pyrenees (endangered species) (Cardona et al. 2010), and the zebra mussel in Ribarroja reservoir (exotic invasive species) (Cardona et al. 2011). These works have led to a formal, computational modeling framework called Population Dynamics P systems (PDP systems) (Colomer et al. 2013).

In order to experimentally validate these P systems based models, the development of simulators is requested (Păun et al. 2010). P-Lingua García-Quismondo et al. (2010), http://www.p-lingua.org is a simulation framework for P systems, which aims to be generic, multi-platform (it is written in Java) and to provide a standard description language for P systems. It has been used to develop simulators for many variants of P systems, specially for PDP systems. Furthermore, experts and model designers are able to run virtual experiments in an abstracted way (without the need of accessing to details of P systems) through a special software called MeCoSim (Pérez-Hurtado et al. 2010, http://www.p-lingua.org/mecosim). MeCoSim uses P-Lingua as the simulation core.

The run times offered by these general simulation frameworks are high for some scenarios involving large and complex models. This lack of efficiency is mainly entailed by the facts of both using Java Virtual Machine and implementing sequential algorithms (Martínez-del-Amor 2013).

Indeed, simulating massively parallel devices like P systems in a sequential fashion is twice inefficient. This issue can be addressed by harnessing the highly parallel architecture within modern processors to map the massively parallelism of P systems (Martínez-del-Amor 2013; Martínez-del-Amor et al. 2015).

Whereas commodity CPUs can contain dozens of processors, current graphic processors (GPUs) (Harris 2005; Owens et al. 2008) provide thousands of computing cores. They can be programmed using general-purpose frameworks such as CUDA (Kirk and Hwu 2010; NVIDIA CUDA website 2015), OpenCL and OpenAcc. GPUs exploit data parallelism by using a very fast memory and simplistic cores. Given the high level of parallelism within modern GPUs (up to 3500 cores per device NVIDIA CUDA website 2015), they have provided a platform to implement real parallelism of P systems in a natural way. Many P system models have been considered to be simulated with CUDA (Martínez-del-Amor et al. 2015): P systems with active membranes, solutions for SAT with families of P systems with active membranes and of tissue P systems with cell division, Enzymatic Numerical P systems, Spiking Neural P systems without delays, and Population Dynamics P systems (Martínez-del-Amor et al. 2012c), among others. Most of these simulators are within the scope of PMCGPU (Parallel simulators for Membrane Computing on the GPU) software project (The PMCGPU project 2013), which aims to gather efforts on parallelizing P system simulators with GPU computing.

As shown by all of these research works, the development of a new P system simulator requires a big research and development effort. For example, in the case of the simulator for PDP systems, the simulation algorithm called DCBA (*Direct distribution based on Consistent Blocks Algorithm*) (Colomer-Cugat et al. 2014; Martínez-del-Amor et al. 2012c) was implemented. It is based on four different phases with completely different characteristics, and the parallelization effort is also different in each one (e.g. second phase of DCBA is a random sequential loop that cannot be easily parallelized). Therefore, the different semantical and syntactical elements of each P system variant lead to completely different GPU-based simulators. Not only does the GPU code depends on the simulated variant, but its efficiency also depend on the simulated P system within the variant (Martínez-del-Amor et al. 2012b).

In this paper, we discuss new developments on the GPU simulator for PDP systems. In summary, a new input module for receiving binary files has been created, allowing to run real ecosystem models defined with P-Lingua. Moreover, we present a road map proposal, a set of research lines for future work that is going to be addressed.

The paper is structured as follows: Sect. 2 provides an overview of the required concepts to understand this paper. Section 3 presents the new feature of the simulator consisting in a input module to read binary files, and also some preliminary results. Section 4 deals with experimental results (validation and performance analysis) for a real ecosystem. Finally, Sect. 5 discuses future developments to take into consideration.

2 Preliminaries

In this section we briefly provide the minimum concepts for the understandability of the paper. We will not formally introduce the model of PDP systems and the concept of GPU computing into detail. Instead, we provide short descriptions along with the corresponding references.

2.1 PDP systems model and simulation: DCBA

PDP systems (Colomer et al. (2013; Colomer-Cugat et al. 2014) constitute a variant of multienvironment P systems (García-Quismondo et al. 2014), which consists in a directed graph whose nodes are called environments. Each environment contains a single cell-like P system. Moreover, the arcs of the graph is implicitly given by a set of communication rules which allow the movement of objects between environments in a one-to-many fashion. Thus, these rules are of the form: $(x)_{e_j} \xrightarrow{p_r} (y_1)_{e_{j_1}} \cdots (y_h)_{e_{j_h}}$, where $x, y_1, \ldots, y_h$ are objects from the environment alphabet. All the P systems within each environment have the same skeleton. In other words, they have the same membrane structure (with three polarizations associated with the membranes), the same working alphabet, and the same set of evolution rules. These rules are of the form: $u\ [\ v\ ]_i^\alpha \rightarrow u'\ [\ v'\ ]_i^\beta$, where u, v, u', v' are multisets over the working alphabet, i a membrane id, and α, β polarizations from the set $\{0, +, -\}$. In some sense, these P systems can be seen as an extension of the active membranes model. However, no dissolution neither division are allowed, and special care on the consistency of rules has to be taken.

PDP systems have also a probabilistic flavor in terms of probabilities associated with the rules. On the one hand, a probability is assigned to each (skeleton) evolution rule for each environment, thus being of the following form: $u\ [\ v\ ]_i^\alpha \xrightarrow{f_{r,j}} u'\ [\ v'\ ]_i^\beta$. On the other hand, a probability function is assigned to each communication rule (see p_r in the above description). Rules are executed in a maximal parallel way according to the probabilities. Rules having the same left-hand side must satisfy the following condition: the summing of their probabilities has to be 1.

Inherently to the model is the concept of rule block: a block is formed by rules having the same left-hand side.

For the syntax of the models, refer to Colomer-Cugat et al. (2014), Colomer et al. (2013) and García-Quismondo et al. (2014). Concerning the semantics of the model, several simulation algorithms have been proposed since the introduction of PDP systems. Each new algorithm aimed at improving the accuracy of mapping the reality to the models. Perhaps, the most difficult feature to handle by the simulation algorithms is the competition of objects between rules from different blocks (note that rules within a block have the same left-hand side, and the objects are consumed according to the probabilities) (Martínez-del-Amor 2013).

The latest introduced algorithm for PDP system is called *Direct distribution based on Consistent Blocks Algorithm (DCBA)* (Martínez-del-Amor et al. 2012c). The approach taken in it is based on the idea of distributing the objects along the rule blocks in a proportional way. After this distribution, the rules within the blocks are selected according to their probabilities using a multinomial distribution. In summary, DCBA consists in 4 phases: 3 for selecting rules and the last one for performing the execution. The scheme of DCBA is the following:

1. Initialization of the algorithm: *static distribution table* (**columns:** blocks, **Rows:** (objects,membrane))
2. **Loop over Time**
3. **Selection** stage:
4. **Phase 1** (Distribution of objects along rule blocks)
5. **Phase 2** (Maximality selection of rule blocks)
6. **Phase 3** (Probabilistic distribution, blocks to rules)
7. **Execution** stage

The proportional distribution of objects along the blocks is carried out through a table which implements the relations between blocks (columns) and objects in membranes (rows). We always start with a static (general) table, and depending on the current configuration of the PDP system, the table is dynamically modified by deleting columns related to non-applicable blocks. Note that after phase 1, we have to assure that the maximality condition still holds. This is normally conveyed by a random loop over the remaining blocks.

Finally, DCBA also handles the consistency of rules by defining the concept of consistent blocks (Martínez-del-Amor et al. 2012c; Martínez-del-Amor 2013): rules within a block have the same left-hand side and the same charge in the right-hand side. There is a further restriction within phase 1: if two non-consistent blocks (having different associated right-hand charge) can be selected in a configuration, the simulation algorithm will return an error, or optionally non-deterministically choose a subset of consistent blocks.

2.2 GPU computing

Today, PC's processors offer from 2 to 16 computing cores, and this number can be increased to 64 or even 128 in high end equipment. These cores are complex enough to run threads simultaneously, each one with its own context, exploiting a coarse grain level of parallelism. For example, OpenMP (http://www.openmp.org) is a threading library for multicore processors, which can be used in C/C++.

High Performance Computing world has changed in the past years. The introduction of the GPU (Harris 2005) as a co-processor unit to compute and render 3-D graphics, encouraged the change of trend in HPC solutions and started to consider heterogeneous platforms having CPUs and co-processors. The GPU has been devised as a highly parallel processor since it was conceived, and now GPGPU enables the GPU to be used for general purpose scientific applications (http://www.gpgpu.org).

A GPU consists of SIMD multiprocessors interconnected to a fast bus with the main memory system (Owens et al. 2008; Kirk and Hwu 2010). Each multiprocessor has a set of computing cores that execute instructions synchronously (they always perform the same instruction over different data) and a small portion of sketchpad memory (similar to caches in CPUs, but manually managed by programmers), among other elements. Current GPUs also implement cache memories (one L2 at the level of the memory system, and a L1 cache which resides within the sketchpad memory).

Fortunately, all these aspects are abstracted to the programmer with high level programming models such as CUDA (Kirk and Hwu 2010; NVIDIA CUDA website 2015). Introduced by NVIDIA in 2007, CUDA allows to run thousands of lightweight *threads* concurrently arranged in *blocks*. Threads belonging to the same block can cooperate and be easily synchronized. Threads from different blocks can only be synchronized by finishing their execution. All these threads execute the same code, called *kernel*, in a SPMD (single-program, multiple-data) fashion, since they access to different pieces of data by using the identifiers associated with each thread and block. Moreover, each thread can also take different branches of execution, but this is penalized when happened within a *warp* (a group of 32 threads), given that it will make the execution to be serialized. The largest but slowest memory system is called global memory, whereas the smallest but fastest sketchpad memory belonging to each block is called shared memory. The access to these memories should be done carefully, since best bandwidth is achieved when threads access to memory in coalesced (to contiguous addresses) and aligned way (Owens et al. 2008).

Finally, the GPU architecture has been improving by the different releases. GT800, Fermi, Kepler and Maxwell are

the codenames of each NVIDIA GPU generation. Each one has been associated with a Compute Capability (CC), 1.X, 2.X, 3.X, and 5.X, respectively (NVIDIA CUDA website 2015).

2.3 PDP systems parallel simulation on the GPU

As mentioned above, the main objective of DCBA is to improve the accuracy of the algorithm. However, it comes at expenses of lower efficiency. Currently, P-Lingua framework implements the algorithm, but it is usually not recommend when dealing with large models because of the large simulation times required. This lack of efficiency is mainly due to the facts of using Java Virtual Machine and implementing sequential algorithms. Indeed, simulating massively parallel devices like P systems in a sequential fashion is twice inefficient. A solution to outcome this issue is by harnessing the highly parallel architecture within modern processors to map the massively parallelism of P systems (Martínez-del-Amor 2013).

GPUs provide a good platform to implement real parallelism of P systems in a natural way, by using their highly level of parallelism (Martínez-del-Amor et al. 2015). Most of P systems simulators based on GPU are within the scope of PMCGPU (*Parallel simulators for Membrane Computing on the GPU*) software project (The PMCGPU project 2013), which aims to gather efforts on parallelizing P system simulators with GPU computing. Specifically, there is a subproject for PDP systems, called ABCD-GPU.

ABCD-GPU started with a multi-core version (Martínez-del-Amor et al. 2012a; Martínez-del-Amor 2013), based on C++ and OpenMP, in which the environments and/or the simulations are distributed along the processors. Experiments showed that parallelizing by simulations leads to better speedups; that is, in a multiprocessor CPU, it is better to parallelize coarsely. In order to deal with finer-grain parallelism, a CUDA version has been also developed (Martínez-del-Amor et al. 2012b; Martínez-del-Amor 2013). In general, these parallel simulators are based on the following principles:

- Efficient representation of the data, both for PDP system syntactical elements and auxiliary structures of DCBA. In this respect, the static and dynamic tables for phase 1 are not really implemented. Instead, the operations over these tables are translated to operations over the syntactical elements of the PDP system, together with much smaller structures. This approach is called virtual table, and has shown to dramatically decrease the required amount of data and time in DCBA.
- Exploiting levels of parallelism presented in the simulation of PDP systems: processing of rule blocks and rules, evolution of environments, and conducting several simulations to extract statistical data from the probabilistic model.

As mentioned in previous section 2.2, CUDA requires a large amount of parallelism to effectively use GPUs resources (Kirk and Hwu 2010). Parallelizing only by simulations as in the OpenMP version is not enough, and the parallelism level is coarse. Instead, the solution was to extract more parallelism from PDP systems as follows (Martínez-del-Amor et al. 2012b):

- *Thread blocks* they are assigned to each environment and each simulation. For each transition step, there is a minimal communication along environments (only when executing communication rules), and each simulation can be executed independently.
- *Threads* each thread is assigned to each rule block/column in selection phases (1, 2 and 3). In execution phase (4), threads will execute rules in parallel. As it is possible to have more rule blocks than threads per thread block, they perform a loop over rule blocks in tiles.

This design is normally tight to the simulated model (depends on the number of environments and blocks) and to the user (number of desired simulations), but it would allow to launch enough CUDA threads and warps to hide memory latencies (Kirk and Hwu 2010). Note that it could be difficult to split rule blocks also to different thread blocks, since we need to synchronize the consumption of objects for phases 1 and 2, and this is allowed only within thread blocks in CUDA.

Furthermore, a special implementation was required for phases 2 and 3, as summarized next (see Martínez-del-Amor et al. 2012b; Martínez-del-Amor 2013 for details):

- Phase 2: Since this phase is inherently sequential, the first approach was a random loop over the remaining blocks, loosing in this way parallelism within thread blocks. The second approach was to dynamically check which blocks compete with each other, in a $O(n)$ loop which tests this condition in shared memory. Once the algorithms indicate which blocks compete for objects, it can assign directly the maximal applications in parallel (for each non-competing group of blocks).
- Phase 3: CUDA comes along with a set of scientific libraries that ease the development of new algorithms. Specifically, there is a simple and parallel library for pseudorandom and quasirandom numbers generation which provides implementation for uniform, normal and poisson distributions. However, the simulator requires multinomial random numbers. In this phase, a new library called CURNG_BINOMIAL was implemented, which generates binomial random variates in

two ways: using the algorithm called BINV for low values of $n \cdot p$ (for a binomial $B(n,p)$), or using a normal approximation otherwise.

So far, ABCD-GPU simulator has been tested by using randomly generated PDP systems. The goal was to provide a flexible way to construct benchmarks for performance analysis, by stressing the simulator with different topologies. In Martínez-del-Amor et al. (2012b), the performance of the simulator was tested with PDP systems having different lengths of the left-hand sides (in terms of number of different objects in the multisets u and v), in average. Running on a NVIDIA Tesla C1060 GPU (which has 240 cores and CC 1.3), these results clearly showed that phase 2 is the bottleneck of the simulator, since it is the less parallel phase. Moreover, when the competition for objects increase (having more objects in the LHS leads to more competitions), the overall performance drastically decreases. Finally, the achieved speedup was of up to $7\times$.

3 A new input module: binary files

After the first version of ABCD-GPU (Martínez-del-Amor et al. 2012b), the efforts were focused on creating a input module to read PDP system descriptions. In this section, we briefly present the new features of the ABCD-GPU simulator, which is a module to read binary files defining PDP systems models. We will also show the first results of the simulator with a real ecosystem model in next section.

3.1 Format definition

Similarly to the simulator of P systems with active membranes (Martínez-del-Amor 2013; Martínez-del-Amor et al. 2015), the design decision for the input file was a binary format. The reason for this is twofold:

- *Size of files* the GPU simulator is conceived for running very large models. Otherwise, it is not worth to be used. Thus, the communication with the simulator should be as efficient as possible to avoid overheads. Since we use P-Lingua for describing PDP system models, it makes sense to use pLinguaCore to parse the files. In this respect, P-Lingua is used as the parser and compiler which send a file to the simulator with unwrapped rules (recall that rules in P-Lingua can be defined in a symbolic way). Thus, in order to reduce the size of the file as much as possible, we have defined a binary format which assign the less amount of bits to each syntactic element.
- *Efficiency* related with the latter, the binary file is also organized in such a way that it fits well with the initialization of structures in the simulator. This helps the efficiency of the parser, while reducing the size of the files.

Although using this kind of format leads to a coupled design (between the P-Lingua parser and the simulator), this will allow to use the GPU engine while reducing the communication/storage cost.

Next, we show the structure of the format for the binary file, which is divided into five sections. The aim is to distribute the information in such an order that the simulator can start allocating space for internal data structures "on the fly".

- *Header* unequivocally identify this file as a binary description file for PDP systems.
- *Sub-header* defines the accuracy used along the file, for the different fields. This allows to use the exact number of bytes according to the number of objects, rules, etc.
- *Global sizes* define the size of alphabet, number of rules, membranes, environments and membrane structure.
- *Rule blocks* their information is given in three subsections, each one giving information for allocating space related with the next one.
- Initial configuration description.

The detailed structure of the binary file is available as a text file since version 1.0 of ABCD-GPU project at PMCGPU (The PMCGPU project 2013).

3.2 Input/output parsers

The ABCD-GPU simulator has been extended with an input module which is able to read files with the above described binary format. Currently, the version is still experimental, and in order to decouple the input parser from the simulator structures, the module creates extra temporal data structures. In the final version, these structures will be avoided, making the reading of input files much more efficient. The input PDP systems can be used both by the CPU and the GPU simulators within the ABCD-GPU platform.

On the other side, a first output module has been also developed. So far, the results were printed on screen just for debugging purposes. Today, it is possible to generate CSV (Comma Separated Values) files, which can be opened by statistics software such as R, and office tools like Excel and Calc. Outputs modules for binary files and data bases interconnection are still under development, and might come with future versions.

Last but not less, P-Lingua has been also extended with a new compiler option for generating binaries from PDP systems. In Cecilia et al. (2010), P-Lingua was first time extended with a compiler for binary files, in which binaries

were generated for P systems with active membrane models. In this case, similar ideas have been used to generate PDP systems binaries. This new compiler will be included in future releases of pLinguaCore. However, a java binary (jar) file is attached to ABCD-GPU since version 1.0, in order to allow the generation of input files easily.

4 Experimental results with a real ecosystem model

The new input module for binary files has enabled to run tests on this simulator with real ecosystem models. In this paper, the model of the Bearded Vulture ecosystem in the Pyrenees, presented in Cardona et al. (2010), was chosen for its simplicity, allowing to carry out debugging and performance testing, and experimental validation.

4.1 Experimental validation

To carry out the experimental validation, 100 simulations were performed for 42 time steps (as required by the model for 14 years, and 3 steps for each cycle), by comparing results for the sequential simulator for DCBA implemented in P-Lingua (already validated) with our simulator running on GPU. Population distribution of Bearded Vulture for the 14 simulated years are shown in Tables 1 and 2. Results are sound, hence we can deduce that our simulator is working adequately.

4.2 Performance analysis

One benchmark has been carried out, running 1000 simulations for 42 time steps (as required by the model for 14 years, and 3 steps for each cycle), and using two GPUs from different generations: (a) Tesla C1060 (GT800 architecture, 240 cores at 1.3 Ghz, 4 GBytes of memory at 800 Mhz and 512-bit bus width, no cache memories), and (b) Tesla K40 (Kepler architecture, 2880 cores at 0.75 Ghz, 12 Gbytes of memory at 3 Ghz and 384-bit bus width, 1.5 MB L2 memory). Table 3 shows the results extracted from both GPU and CPU simulators.

Table 1 Population distribution of the Bearded Vulture—CPU

Year	Average	Deviation
1	21	3
2	22	3
3	23	4
4	25	5
5	26	5
6	27	6
7	28	7
8	30	7
9	32	8
10	33	10
11	34	10
12	36	11
13	38	12
14	40	14

Table 2 Population distribution of the Bearded Vulture—GPU

Year	Average	Deviation
1	21	2
2	22	4
3	23	4
4	24	5
5	25	6
6	27	7
7	28	7
8	29	8
9	30	9
10	32	10
11	33	11
12	35	12
13	36	13
14	38	14

The results show that the K40 GPU achieves better performance, up to 18.1× of speedup with respect to the sequential version, while the Tesla C1060 achieves barely 4.9×. As shown before, Tesla C1060 can achieve up to 7× of speedup with randomly generated P systems. However, there are two new behaviors that were not present before:

- Phase 2 is not the bottleneck. Indeed, it is easy to see that the considered model has no competition for objects. Thus, phase 2 is not required for this simulation, although the only mechanism carried out is the checking of remaining active blocks.
- Phase 1 is the bottleneck for both GPUs, as for the CPU. This phase consists in fact in the execution of three different kernels, having therefore three global synchronization points. However, phase 1 is well accelerated on K40, with 18×, and 10 times faster than C1060.
- Phase 4 is the second slowest phase in Tesla C1060. Since a few ratio of rules is executed, the generation of objects is not performed completely efficiently. The main reason is the usage of atomic operations for adding new objects.
- Phase 3, where the random number generation takes place, is well accelerated in both GPUs. This is the most intensive computing phase in the simulation process, and so, GPUs are faster.

Table 3 Profiling performance with the Bearded Vulture ecosystem model (2008)

	Tesla C1060			Tesla K40		
	% CPU1	% GPU	Acc	% GPU	Acc (CPU2)	Acc (C1060)
Phase 1	79.7	89.4	4.4×	79.4	18.1×	10×
Phase 2	2.4	1.6	7.6×	5.5	7.9×	2.5×
Phase 3	11.8	3.8	14.9×	10.3	19.1×	3.4×
Phase 4	6.1	5.2	5.8×	4.8	28.2×	9.6×
Total			4.9×		18.1×	9×

CPU1 is a Intel Xeon E5504 at 2 Ghz, and CPU2 is a Intel Xeon E3-1230 at 3.3 Ghz. They master, on different machines, Tesla C1060 and Tesla K40 GPUs respectively

In Table 3, it is also possible to see that K40 GPU achieves better performance results than its predecessor C1060, as expected. Specifically, phase 1 and 4 are up to 10 times faster than in C1060. These phases are data intensive, because it requires the upload (and download respectively) of object multiplicities from (to) the global memory system. L2 and L1 cache memories, and also the higher memory bandwidth, have improved the run time for these phases. This result shows that better bandwidth and L2 cache help to accelerate the simulation of PDP systems, since this task is memory bandwidth bounded (Martínez-del-Amor 2013).

5 Road map

Figure 1 shows the current structure of the ABCD-GPU project. The simulation engine implements DCBA in both multicore (CPU) and manycore (GPU) platforms. The input files are generated by pLinguaCore, which acts as a parser in the creation of binary files. The output files will be both in CSV and binary formats soon, and a module to upload results to a database is also still under consideration. Moreover, the platform still support the input of randomly generated PDP systems and the output of corresponding profiling and debugging information, in order to conduct performance benchmarks in future versions of the simulator.

As mentioned, ABCD-GPU project is still under development. The results discussed above belong to the latest version, but improvements for both accuracy and efficiency are planned. Next, the list of future research lines are presented.

1. The retrieval of information from the GPU is still not made efficiently, since the current version has just validation purposes. For example, it is possible to asynchronously copy data from GPU to CPU with

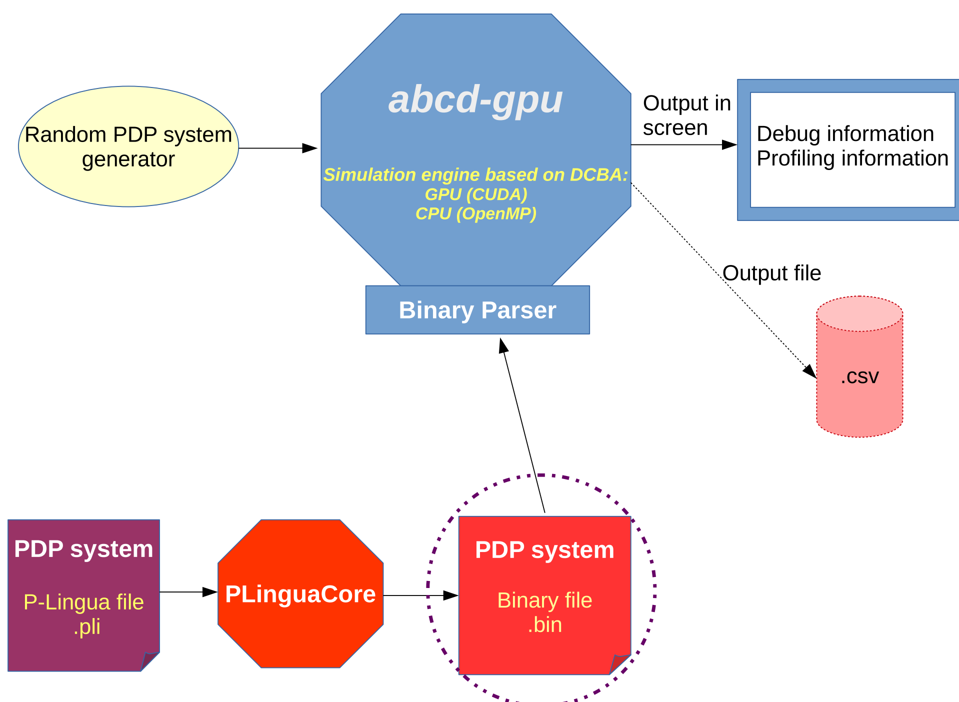

Fig. 1 Structure of ABCD-GPU project

pinned memory. Although it would require a double buffer for the multisets, the transaction of PDP system configurations can be made in parallel to the simulation of the next step. Another idea is to filter the multisets on the GPU, according to some parameters defined by the user.

2. The output module shall be finished for binary files, along with the development of a module that uploads the results into a database (interoperability with MeCoSim framework) in parallel with the simulation.
3. Phase 2 can be slow for models with high rate of object competition, as seen with randomly generated PDP systems. Some ideas in this respect are: (a) Auto-detect if phase 2 is really required (by checking if the model has no competition of objects, or if no active blocks remain after Phase 1), (b) compact active blocks after phase 1 for more efficiency, and (c) impose a (real) random disorder of rule blocks (maybe taking some ideas from Gastalver-Rubio 2012).
4. Avoid current synchronization of DCBA phases, specially for phase 1. That is, run all the phases with one single kernel (perhaps one global kernel which calls to __device__ versions of current kernels). It could be convenient to keep current version (separated kernels) for GPUs that are used by the graphic system on the computer, because of the limitation of kernel time (normally up to 7 s).
5. In PDP systems, the working alphabets for the skeleton and for the environments are disjoint sets (García-Quismondo et al. 2014). Therefore, we can work with all the communication rules apart from the virtual table.
6. Implement a variant of DCBA, called μ DCBA. The aim of this variant is to extract more parallelism within each environment. If we pre-calculate the group of rules that really depend on each other because they compete for objects, we will be able to apply DCBA separately to each group, i.e. more locally and in parallel. Moreover, there will be less resources to handle (and perhaps we would be able to move more data into shared memory, such as the multisets). We define a transitive relation between rule blocks, called *competition*: block b_i *directly* compete for objects with block b_j if they have overlapping but not equal left-hand side. Moreover, if b_k directly compete with B_j, but not with B_i, then B_i and B_k also compete for objects (however, indirectly through B_j). Before starting the simulation, we could apply this algorithm to define disjoint sets of rule blocks holding the competition relation, and then apply DCBA locally to each one (synchronized in each computational step). Finally, it would be desirable to use μDCBA only when the sets are balanced. We could also assign different "small" sets to one thread block.
7. Implement model-oriented optimizations. That is, analyze the PDP system model prior to the simulation and extract properties that will help to the efficiency. For example, test if there is competition for objects, inconsistent rule blocks, etc.
8. *Parallel P-Lingua*: it would be interesting to let the model designer to provide the above mentioned properties to the simulator. For example, to allow in P-Lingua the usage of directives for defining modules of rules that can be executed in parallel, similarly to the `pragma` directives in OpenMP.
9. Hybrid simulation of PDP systems, by using both the CPU and GPU platforms at the same time, and implement a merge module of simulations at the end of the process.

6 Conclusions

In this paper, new version of the CPU/GPU simulators for PDP systems (ABCD-GPU project) is presented. This version (1.0 beta) of ABCD-GPU can be downloaded from PMCGPU website. The new feature is a input module which supports files with a binary format, aiming to compress the information provided in the models. The purpose of using a restricted, binary format is for efficiency in its communication. Moreover, this input module has allowed to run tests with real ecosystem models. Specifically, we have chosen the ecosystem of the Bearded Vulture in the Catalan Pyrenees presented in Cardona et al. (2010), in order to experimentally validate the simulator. Furthermore, performance analysis has been made using a Tesla C1060 and a Tesla K40 GPUs.

The results show that with this real ecosystem model, phase 1 of DCBA is the bottleneck, since there is no competing blocks, what effectively disable phase 2 in the simulation. Moreover, we have shown that next generation GPUs, such as K40, achieves better performance, given their higher memory bandwidth and their L2 caches. For example, phases 1 and 4, which are the most data intensive in DCBA, are 10 times faster in K40 than in its predecessor, demonstrating our theory that P system simulations are memory bandwidth bounded.

Finally, it is worth to mention that, in this case, Parallel Computing is not only used to get faster solutions, but also, to obtain better results, because it will enable the users to run DCBA-based simulations in an affordable time.

Acknowledgments The authors acknowledge the support of the project TIN2012-37434 of the "Ministerio de Economía y Competitividad" of Spain, co-financed by FEDER funds. They also acknowledge the CUDA Research Center program, granted by NVIDIA to the University of Seville in 2014, 2015 and 2016, and

their donation of a Tesla K40 GPU. Finally, Martínez-del-Amor also acknowledges the support of the 3rd postdoctoral phase of the PIF program associated with the project of excellence from "Junta de Andalucía" under grant P08–TIC04200, co-financed by FEDER funds.

References

Cardona M, Colomer MA, Margalida A, Pérez-Hurtado I, Pérez-Jiménez MJ, Sanuy D (2010) A P system based model of an ecosystem of some scavenger birds. LNCS 5957:182–195. doi:10.1007/978-3-642-11467-0_14

Cardona M, Colomer MA, Margalida A, Palau A, Pérez-Hurtado I, Pérez-Jiménez MJ, Sanuy D (2011) A computational modeling for real ecosystems based on P systems. Nat Comput 10(1):39–53. doi:10.1007/s11047-010-9191-3

Cecilia JM, García JM, Guerrero GD, Martínez-del-Amor MA, Pérez-Hurtado I, Pérez-Jiménez MJ (2010) Simulation of P systems with active membranes on CUDA. Brief Bioinform 11(3):313–322. doi:10.1093/bib/bbp064

Colomer MA, Margalida A, Pérez-Jiménez MJ (2013) Population dynamics P system (PDP) models: a standardized protocol for describing and applying novel bio-inspired computing tools. PLoS One 8(4):e60698. doi:10.1371/journal.pone.0060698

Colomer-Cugat MA, García-Quismondo M, Macías-Ramos LF, Martínez-del-Amor MA, Pérez-Hurtado I, Pérez-Jiménez MJ, Riscos-Núñez A, Valencia-Cabrera L (2014) Membrane system-based models for specifying dynamical population systems. In: Frisco P et al (eds) Applications of membrane computing in systems and synthetic biology. Emergence, complexity and computation series, chap 4, vol 7. Springer International Publishing, Switzerland, pp 97–132

García-Quismondo M, Gutiérrez-Escudero R, Pérez-Hurtado I, Pérez-Jiménez MJ, Riscos-Núñez A (2010) An overview of P-Lingua 2.0. LNCS 5957:264–288. doi:10.1007/978-3-642-11467-0_20

García-Quismondo M, Martínez-del-Amor MA, Pérez-Jiménez MJ (2014) Probabilistic guarded P systems: a new formal modelling framework. LNCS 8961:194–214. doi:10.1007/978-3-319-14370-5_12

Gastalver-Rubio A (2012) Simulation of probabilistic P systems on GPUs. Final Research Project, University of Seville

GPGPU organization. http://www.gpgpu.org

Harris M (2005) Mapping computational concepts to GPUs. In: ACM SIGGRAPH 2005 Courses, New York

Kirk D, Hwu W (2010) Programming massively parallel processors: a hands on approach. Morgan Kaufmann, Waltham

Martínez-del-Amor MA (2013) Accelerating membrane systems simulators using high performance computing with GPU, Ph.D. thesis, University of Seville

Martínez-del-Amor MA, Karlin I, Jensen RE, Pérez-Jiménez MJ, Elster AC (2012a) Parallel simulation of probabilistic P systems on multicore platforms. In: García-Quismondo M et al (eds.) Tenth brainstorming week on membrane computing, vol II. Fénix editora, Sevilla, pp 17–26

Martínez-del-Amor MA, Pérez-Hurtado I, Gastalver-Rubio A, Elster AC, Pérez-Jiménez MJ (2012b) Population dynamics P systems on CUDA. In: 10th Conference on computational methods in systems biology, LNBI, vol 7605, pp 247–266

Martínez-del-Amor MA, Pérez-Hurtado I, García-Quismondo M, Macías-Ramos LF, Valencia-Cabrera L, Romero-Jiménez A, Graciani C, Riscos-Núñez A, Colomer MA, Pérez-Jiménez MJ (2012c) DCBA: simulating population dynamics P systems with proportional object distribution. LNCS 7762:27–56. doi:10.1007/978-3-642-36751-9_18

Martínez-del-Amor MA, García-Quismondo M, Macías-Ramos LF, Valencia-Cabrera L, Riscos-Núñez A, Pérez-Jiménez MJ (2015) Simulating P systems on GPU devices: a survey. Fundam Inform 136(3):269–284. doi:10.3233/FI-2015-1157

NVIDIA CUDA website (2015). https://developer.nvidia.com/cuda-zone

OpenMP webiste. http://www.openmp.org

Owens JD, Houston M, Luebke D, Green S, Stone JE, Phillips JC (2008) GPU computing. Proc IEEE 96(5):879–899

Păun G (2000) Computing with membranes. J Comput Syst Sci 61(1):108–143. doi:10.1006/jcss.1999.1693

Păun G, Rozenberg G, Salomaa A (eds) (2010) The Oxford handbook of membrane computing. Oxford University Press, Oxford

Pérez-Hurtado I, Valencia-Cabrera L, Pérez-Jiménez MJ, Colomer MA, Riscos-Núñez A (2010) MeCoSim: a general purpose software tool for simulating biological phenomena by means of P systems. In: Proceedings IEEE fifth international conference on bio-inpired computing: theories and applications (BIC-TA 2010), vol I, pp 637–643. doi:10.1109/BICTA.2010.5645199

Pérez-Jiménez MJ, Romero-Campero FJ (2006) P systems, a new computational modelling tool for Systems Biology. Trans Comput Syst Biol VI LNBI 4220:176–197. doi:10.1007/11880646_8

Romero-Campero FJ, Pérez-Jiménez MJ (2008) A model of the quorum sensing system in Vibrio fischeri using P systems. Artif Life 14(1):95–109. doi:10.1162/artl.2008.14.1.95

The MeCoSim web page. http://www.p-lingua.org/mecosim

The P-Lingua web page. http://www.p-lingua.org

The PMCGPU project (2013) http://sourceforge.net/p/pmcgpu