Extracting Parallelism in Simulation Algorithms for PDP systems

Miguel Á. Martínez-del-Amor, Andrés Doncel-Ramírez, David Orellana-Martín, Ignacio Pérez-Hurtado

Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
Universidad de Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
E-mail: mdelamor@us.es, andrestp95@gmail.com, dorellana@us.es, perezh@us.es

Summary. Population Dynamics P systems is a modelling framework that have been used successfully for some important real ecosystems. This model is inherently probabilistic, and the scheme of rules is very flexible, allowing even cooperation between membranes. Thus, its simulation has been a challenge in the past years, leading to several simulation algorithms. The latest one, which has been proved to be the most accurate so far, is DCBA. The main drawback of DCBA is its complexity, requiring a very large table to handle all competitions. In this paper, we discuss two strategies to decrease this table, allowing a more lightweight version of DCBA that can be used in parallel implementations.

Keywords: Membrane Computing, Population Dynamics, Parallel simulation

1 Introduction

Some very important real ecosystems have been modelled using the formal framework called Population Dynamics P (PDP) systems [8, 6]. This framework consists of a multienvironment P system model [11] that contains one single cell-like P system within the nodes (environments) of a directed graph. Each of the cell-like P systems have the same skeleton (membrane tree and evolution rules). Thus, PDP systems have to kinds of rules: evolution (skeleton) and communication rules. Moreover, PDP systems is a probabilistic model, in the sense that probabilities are associated with the rules. Skeleton rules may have associated different probabilities regarded the environment where they are located.

The very flexible pattern of skeleton rules, where object cooperation can happen even between membranes (the active membrane and its parent), increases the complexity when designing simulation algorithms. For this reason, several

approaches have been made: BBB (Binomial Block Based), DNDP (Direct Non Deterministic distribution with Probabilities) and DCBA (Direct distribution based on Consistent Blocks Algorithm) [13, 3, 5]. These algorithms are designed to tackled specifically the competition for resources that can happen in the models. Specifically, different rules having overlapping but different left-hand sides compete for objects in the multisets.

DCBA is the algorithm that has been demonstrated to show more accurate results, according to the way the formal framework is employed for ecosystem modelling [3]. Moreover, it is highly parallelizable, as shown in the GPU implementation called ABCD-GPU [12, 15]. However, the algorithm has several drawbacks:

1. it consists of four phases to simulate just one computation step;
2. the first phase uses a distribution table that does not scale well when increasing the amount of rules and objects in the alphabet;
3. the second phase is inherently sequential;

In this paper, we discuss two different strategies to cope with the second drawback mentioned above: *adaptative DCBA* and *μ-DCBA* (or DCBA with partitions of rules). The former, already published in [16], is a solution where the designer provides high-level information of the model, such as rule modules, so that the simulator knows that DCBA must be applied locally to each module. The latter is a novel solution, still unpublished, where the rule competition is pre-computed (before starting the simulation), so that DCBA is applied locally to each partition. These two methods were first mentioned in [12], but we extend these ideas here.

The rest of the paper is structured as follows: Section 2 briefly recall the model of PDP systems and Section 3 its definitions for DCBA; Section 4 describes the concept of adaptative simulator and how it is applied to DCBA; Section 5 introduces the idea of μ-DCBA; and finally, Section 6 ends the document with conclusions and future work.

2 Population Dynamics P systems

Next, we recall the formal definition of a PDP system. We also provide some concepts required for DCBA, and the main loop of the algorithm. More information than the one provided here can be found in [7, 9, 3].

Definition 1. *A Population Dynamics P system of degree (q, m) with $q \geq 1$, $m \geq 1$, and taking $T \geq 1$ time units, is a tuple*

$$\Pi = (G, \Gamma, \Sigma, T, R_E, \mu, R, \{f_{r,j} : r \in R, 1 \leq j \leq m\}, \{\mathcal{M}_{ij} : 1 \leq i \leq q, 1 \leq j \leq m\})$$

where:

- $G = (V, S)$ *is a directed graph. Let the vertices be* $V = \{e_1, \ldots, e_m\}$*, also called environments;*
- Γ *is the working alphabet and* $\Sigma \subsetneq \Gamma$ *is an alphabet representing the only objects that can be present in the environments;*
- T *is a natural number that represents the simulation time of the system;*
- R_E *is a finite set containing the so called communication rules, that send objects between environments. They are of the form*

$$(x)_{e_j} \xrightarrow{p} (y_1)_{e_{j_1}} \cdots (y_h)_{e_{j_h}}$$

where $x, y_1, \ldots, y_h \in \Sigma$*,* $(e_j, e_{j_l}) \in S$ $(1 \leq l \leq h)$ *and* p *is a computable function from* $\{1, \ldots, T\}$ *to* $[0, 1]$*. By default and for simplicity, we assume* $p = 1$*, in case it is not specified for a rule. Moreover, for each rule of this form, the following holds:* $(e_j, e_{j_h}) \in S$*. These functions verify the following:*
  - *For each* $e_j \in V$ *and* $x \in \Sigma$*, the sum of functions associated with the rules whose left-hand side is* $(x)_{e_j}$*, is exactly* 1*.*
- μ *is a membrane structure with* q *membranes injectively labelled by* $1, \ldots, q$*. The skin membrane is labelled by 1. An electrical charge from the set* $EC = \{0, +, -\}$ *is also associated with each membrane.*
- R *is a finite set of evolution (skeleton) rules of the form*

$$u[\, v\,]_i^{\alpha} \rightarrow u'[\, v'\,]_i^{\alpha'}$$

where $u, v, u', v' \in \Gamma^*$*,* i $(1 \leq i \leq q)$*,* $u + v \neq \lambda$ *and* $\alpha, \alpha' \in \{0, +, -\}$*. The following restriction must hold:*
  - *If* $(x)_{e_j}$ *is the left-hand side of a rule from* R_E*, then none of the rules of* R *has a left-hand side of the form* $u[v]_1^{\alpha}$*, for any* $u, v \in \Gamma^*$ *and* $\alpha \in \{0, +, -\}$*, having* $x \in u$*.*
- *For each* $r \in R$ *and for each* j $(1 \leq j \leq m)$*, the function* $f_{r,j} : \{1, \ldots, T\} \longrightarrow [0, 1]$ *is computable. These functions verify the following:*
  - *For each* $u, v \in \Gamma^*$*,* i $(1 \leq i \leq q)$*,* $\alpha, \alpha' \in \{0, +, -\}$ *and* j $(1 \leq j \leq m)$ *the sum of functions associated with* j *and the set of rules whose left-hand side is* $u[v]_i^{\alpha}$ *and whose right-hand side has polarization* α'*, is the constant function* 1*.*
- *For each* j*,* $(1 \leq j \leq m)$*,* $\mathcal{M}_{1j}, \ldots, \mathcal{M}_{qj}$ *are strings over* Γ*, describing the multisets of objects initially placed within the regions in environment* e_j *(also known as initial configuration).*

In other words, a PDP system consists of m environments $e_1, \ldots, e_m$ linked between them by the edges from the directed graph G. Each environment e_j contains a P system, $\Pi_j = (\Gamma, \mu, R_{\Pi_j}, \mathcal{M}_{0j}, \ldots \mathcal{M}_{q,j})$, of degree q, where every rule $r \in R$ has a computable function $f_{r,j}$ (specific for environment j) associated with it.

A *configuration* of the system at an instant t is a tuple of multisets of objects present in the m environments and at each of the regions of each Π_j, together with the polarizations of the membranes in each P system. At the initial configuration

of the system we assume that all environments are empty and all membranes have a neutral polarization. As it is usual in cell-like P systems, we also assume that a global clock exists which synchronizes all environments.

The P system can pass from one configuration to the next one by using the rules from $\bigcup_{j=1}^{m} R_{\Pi_j} \cup R_E$ as follows: at each transition step, the rules to be applied are selected according to the *probabilities* assigned to them, and all applicable rules are simultaneously applied in a maximal way (i.e. no more rules can be further applicable). For rules in R_{Π_j}, the charge of the (active) membrane will be changed. In this sense, the consistency of charges must hold: in order to apply several rules of R_{Π_j} simultaneously to the same membrane, all the rules must have the same electrical charge on their right-hand side.

When a communication rule $(x)_{e_j} \xrightarrow{p} (y_1)_{e_{j_1}} \dots (y_h)_{e_{j_h}}$ between environments is applied, object x passes from e_j to $e_{j_1}, \dots, e_{j_h}$ possibly modified into objects $y_1, \dots, y_h$ respectively. At any moment t $(1 \leq t \leq T)$ for each object x in environment e_j, if there exist communication rules whose left-hand side is $(x)_{e_j}$, then one of these rules will be applied. If more than one communication rule can be applied to an object, the system selects one randomly, according to their probability which is given by $p(t)$.

3 Simulation algorithms

The simulation algorithms for PDP systems called BBB and DCBA are based on the grouping of rules into blocks. These groups are constructed by looking the left-hand side. Note that rules having the same left-hand side must have associated probabilities summing 1. Specifically, DCBA works using a refined definition of block, called consistent block, as shown in Definition 2. DNDP does not use the concept of blocks, but it selects rules by a random loop instead.

Definition 2. *Rules from R and R_E are classified into consistent blocks by either of the following:*

a. the rule block associated with $(i, \alpha, \alpha', u, v)$ is $B_{i,\alpha,\alpha',u,v} = \{r \in R : LHS(r) = (i, \alpha, u, v) \wedge charge(RHS(r)) = \alpha'\}$;
b. the rule block associated with (e_j, x) is $B_{e_j,x} = \{r \in R_E : LHS(r) = (e_j, x)\}$.

It is important to remark that the selection of rules in BBB and DCBA relies always first on selecting blocks, calculating a multinomial random variate, and therefore obtaining a selection of rules within each block. In this sense, we can say that rules within a block will not compete among objects when using BBB and DCBA, because they are selected altogether. This, again, does not hold in DNDP, where rules are selected individually according to the probabilities. Block competition will be defined later in Definition 3.

DCBA tackles the resource competition issue by performing a proportional distribution of objects among competing blocks. This is done by using the *distribution table*, which is a system-wide time having blocks per columns, and

pairs (object,region) per rows. Algorithm 1 shows a summary of the algorithm, which can be depicted in [3]. It can be seen that, as usual, each loop iteration is made by two stages: selection and execution. Selection stage consists of three phases: Phase 1 distributes objects to the blocks in a certain proportional way, Phase 2 ensures *maximality* by checking the maximal number of applications of each block, and Phase 3 translates from block to rule applications by calculating random numbers using a multinomial distribution. Finally, execution stage (or Phase 4) generates the right-hand side of rules.

Algorithm 1 DCBA MAIN LOOP

Require: A PDP system Π of degree (q, m), $T \geq 1$ (time units), $A \geq 1$ (*accuracy* parameter), and an initial configuration C_0.

1: $(\mathcal{T}) \leftarrow$ *INITIALIZATION* (Π) ▷ (Initializes the distribution table)
2: **for** $t \leftarrow 0$ **to** $T-1$ **do** ▷ (For each transition step)
3: **SELECTION:** ▷ (Selection of rules, subtracting their left-hand sides)
4: $B_{sel} \leftarrow \emptyset$, $R_{sel} \leftarrow \emptyset$
5: $(\mathcal{T}^t, C'_t, B_{sel}) \leftarrow$ *PHASE 1* $(\Pi, A, C_t, \mathcal{T})$ ▷ (Distribution of objects)
6: $(C'_t, B_{sel}) \leftarrow$ *PHASE 2* $(\Pi, C'_t, B_{sel}, \mathcal{T}^t)$ ▷ (Ensure Maximality)
7: $(R_{sel}) \leftarrow$ *PHASE 3* (Π, B_{sel}) ▷ (Probabilistic distribution)
8: **EXECUTION:** ▷ (Execution of rules, adding their right-hand sides)
9: $(C_{t+1}) \leftarrow$ *PHASE 4* (Π, C'_t, R_{sel})
10: **end for**

It is important to remark that the stages of DCBA can be performed independently (and hence, in parallel) to each environment [14, 15]. However, we need a synchronization point between selection and execution stages, because communication rules might generate objects in different environments. This way, the main loop of DCBA can be rewritten to the form in Algorithm 2.

We can finally identify two main bottlenecks in the simulation algorithms. The first one is tackled by the adaptative

1. All algorithms (DCBA, BBB and DNDP) need to go through every defined rule in the system at every transition step, in order to check if it is applicable. Indeed, there is no way to know in advance which rules can be applicable in each time step.
2. DCBA is specifically designed to cope with block competitions in an accurate way, but it has to assume that all rules can have cross competitions (rule a competes with rule b, and rule b with rule c, then rules a, b and c must agree on the objects to consume).

4 Adaptative DCBA

The idea of adaptative simulators was introduced and analysed in [16]. It is inspired in the way directives work in common programming languages. They are special

Algorithm 2 DCBA MAIN LOOP FOR ENVIRONMENTS

Require: A PDP system Π of degree (q, m), $T \geq 1$ (time units), $A \geq 1$ (*accuracy* parameter), and an initial configuration C_0.

for $j \leftarrow 1$ **to** m **do** ▷ (For each environment j)
  $(\mathcal{T}_j) \leftarrow$ *INITIALIZATION (j,Π)* ▷ (Initializes the table for environment j)
end for
for $t \leftarrow 0$ **to** $T-1$ **do** ▷ (For each transition step)
  for $j \leftarrow 1$ **to** m **do** ▷ (For each environment j)
    SELECTION: ▷ (Selection of rules for environment j)
    $B^j_{sel} \leftarrow \emptyset$, $R^j_{sel} \leftarrow \emptyset$
    $(\mathcal{T}^t_j, C'_t, B^j_{sel}) \leftarrow$ *PHASE 1* $(j, \Pi, A, C_t, \mathcal{T}_j)$ ▷ (Distribution of objects)
    $(C'_t, B^j_{sel}) \leftarrow$ *PHASE 2* $(j, \Pi, C'_t, B^j_{sel}, \mathcal{T}^t_j)$ ▷ (Ensure Maximality)
    $(R^j_{sel}) \leftarrow$ *PHASE 3* (j, Π, B^j_{sel}) ▷ (Probabilistic distribution)
  end for
  for $j \leftarrow 1$ **to** m **do** ▷ (For each environment j)
    EXECUTION: ▷ (Execution of rules for environment j)
    $(C_{t+1}) \leftarrow$ *PHASE 4* $(j, \Pi, C'_t, R^j_{sel})$
  end for
end for

syntactic elements that tell extra information to the compiler, allowing to better adapt the code for some purposes if the compiler accepts it (e.g. in OpenMP, one call easily ask to parallelize the iterations of a loop), or just processing the code dismissing that information (the code is still valid without the directives). This way, a P system model designer can also provide very useful information to the simulator, rather than just the syntactic and/or semantic elements of the P system to simulate. For instance, P system models are usually designed bearing in mind a global algorithmic scheme, where the computation of the P system is subdivided into stages of specific purposes (e.g. in SAT solutions for active membranes, there are stages for generating membranes, other stages for check-in solutions, etc.).

Specifically in PDP systems, ecosystem modellers often use algorithmic schemes for their models [7]. This is done by first defining a cycle, which corresponds to a certain time in the simulated ecosystem (e.g. one year). A cycle in the model is a fixed amount of transition steps where a sequence of modules take place. These modules reproduces certain processes such as reproduction of species, feeding, migration, etc. Moreover, these modules consist of certain rules that are carefully designed to model the corresponding process. Therefore, we can say that somehow, the model designer already knows which rules can be executed in each time step. Thus, if they are able to provide that information in form of a directive-like syntax, the simulator can take advantage of this to dismiss rules automatically at each step.

In [16], the ABCD-GPU simulator was turned into adaptative. First, the model designer is able to provide the information of the modules they are defining by using the new P-Lingua 5 software [17]. This new version now includes new syntax elements called *features*. They are written as `@featureName = featureValue`, and

can be defined globally (for the whole system) or locally (for individual rules). ABCD-GPU takes this information to organize the rules by modules. Of course, if the simulator does not recognize the information provided by the features, it can proceed and simulate the system without problems.

The simulator also pre-computes which modules are active in each step within the cycle, so that it can easily access the rules that might be applicable at each transition step. Furthermore, a parallel implementation can harness this to reach more parallelism, specifically between parallel modules that can be active at certain steps. This design helped to improve the performance by 2.5x extra when using a P100 GPU [16].

As for environments, DCBA's stages can be performed independently per active module, but it requires a synchronization point. Algorithm 3 shows how it can be re-defined to handle modules and environments. There two variables for steps: t is the global time step of the simulation, and s is the step within a cycle. After reading the information provided by the model designer along with the PDP system (e.g. with P-Lingua 5), we get a map to associate rules to each module, and another to know which modules are active in each step of the cycle. Finally, we also provide the total amount of modules d.

5 DCBA with partitions of rules

As mentioned above, DCBA assumes that all blocks can compete for objects. These competitions can make dependencies between blocks that are encoded in the transition table, which takes care of distributing the resources (objects) to the blocks that can be applied. Later on, rules within blocks will compute its applications using a multinomial random variate. In order to decrease the size of the distribution table, we can pre-calculate which blocks are actually competing for resources one each other. This kind of problems have been already tackled in the literature [1, 19]. Thus, in this paper we propose a similar concept, but adapted for PDP systems, where rules have a more flexible pattern. If we focus in DCBA, we can formally define the condition of block competition as shown in Definition 3.

Definition 3. *Two consistent blocks* $B^1_{i_1,\alpha_1,\alpha'_1,u_1,v_1}$ *and* $B^2_{i_2,\alpha_2,\alpha'_2,u_2,v_2}$ *compete for objects when both the following holds:*

(a) The two blocks are mutually consistent. That is, if $i_1 = i_2 \wedge \alpha_1 = \alpha_2$ *then* $\alpha'_1 = \alpha'_2$*;*

(b) Their left-hand sides overlap. That is, either of the following conditions hold:

- *If* $i_1 = i_2$ *and* $\alpha_1 = \alpha_2$ *then* $u_1 \cap u_2 \neq \emptyset$ *or* $v_1 \cap v_2 \neq \emptyset$*;*
- *If* $i_1 \neq i_2$ *but* i_1 *is the parent membrane of* i_2*, then* $v_2 \cap u_1 \neq \emptyset$*, or* i_2 *is the parent membrane of* i_1*, then* $u_2 \cap v_1 \neq \emptyset$*.*

It is important to remark that blocks from communication rules do not compete with each other, nor with blocks from evolution rules (see the definition in

Algorithm 3 DCBA MAIN LOOP FOR ENVIRONMENTS AND MODULES

Require: A PDP system Π of degree (q, m), $T \geq 1$ (time units), $A \geq 1$ (*accuracy* parameter), an initial configuration C_0, the number of time units per cycle c, the amount of modules d, a structure mapping the rules and blocks per module MR, and another structure mapping which module is active at each step in the cycle MS.

1: **for** $j \leftarrow 1$ **to** m **do** ▷ (For each environment j)
2: **for** $k \leftarrow 1$ **to** d **do** ▷ (For each module k)
3: $(\mathcal{T}_{j,k}) \leftarrow$ *INITIALIZATION (j,Π,k,MR)* ▷ (Creates the table for environment j and module k)
4: **end for**
5: **end for**
6: $t \leftarrow 0$
7: **while** $t < T$ **do** ▷ (For each transition step)
8: **for** $t \leftarrow t$ **to** $t + c - 1$ **do** ▷ (Looping the transition steps inside a cycle)
9: $s \leftarrow t \; mod \; c$ ▷ (The step within the cycle)
10: **for** $j \leftarrow 1$ **to** m **do** ▷ (For each environment j)
11: **for** $k \leftarrow 1$ **to** d **do** ▷ (For each module k)
12: **SELECTION:** ▷ (Selection for environment j and module k)
13: **if** $MS[k, s]$ **then** ▷ (If module k is active in step s)
14: $B^{j,k}_{sel} \leftarrow \emptyset$, $R^{j,k}_{sel} \leftarrow \emptyset$
15: $(\mathcal{T}^t_{j,k}, C'_t, B^{j,k}_{sel}) \leftarrow$ *PHASE 1* $(j, \Pi, A, C_t, \mathcal{T}_{j,k}, k, MR)$
16: $(C'_t, B^{j,k}_{sel}) \leftarrow$ *PHASE 2* $(j, \Pi, C'_t, B^{j,k}_{sel}, \mathcal{T}^t_{j,k}, k, MR)$
17: $(R^{j,k}_{sel}) \leftarrow$ *PHASE 3* $(j, \Pi, B^{j,k}_{sel}, k, MR)$
18: **end if**
19: **end for**
20: **end for**
21: **for** $j \leftarrow 1$ **to** m **do** ▷ (For each environment j)
22: **for** $k \leftarrow 1$ **to** d **do** ▷ (For each module k)
23: **EXECUTION:** ▷ (Execution for environment j and module k)
24: **if** $MS[k, s]$ **then** ▷ (If module k is active in step s)
25: $(C_{t+1}) \leftarrow$ *PHASE 4* $(j, \Pi, C'_t, R^{j,k}_{sel}, k, MR)$
26: **end if**
27: **end for**
28: **end for**
29: **end for**
30: **end while**

Section 2). Let us represent the competition relationship as an undirected graph $G_c = (V_c, E_c)$, where V_c is the set of all rule blocks and E is the set of edges connecting the blocks that directly compete one with each other. We will therefore say that two blocks will compete, directly or indirectly, if there exists a path between them. Thus, we can calculate partitions of competitions from the set of rule blocks as depicted in Definition 4.

Definition 4. *Given a set of rule blocks* $V = \{B_1, \ldots, B_k\}$*, a partition of competition is a partition of the set* V*,* $P = \{P_1, \ldots, P_l\}$*, where the following holds:*

a block B_i belongs to the set P_i if and only if it competes, directly or indirectly, with the rest of blocks in C_i, and do not compete with any of the rule blocks form the rest of sets in P. The union of the sets in P is V.

Specifically, communication rule blocks form partitions with just one element. It is easy to compute the partitions of competitions from the set of rule blocks by calculating the connected components in the graph G_c. After having this, we can redefine the DCBA algorithm to be executed locally to each partition, if it contains more than one elements (also known as μ-DCBA). Generally, we can re-structure the algorithm as shown in Algorithm 4.

Algorithm 4 DCBA MAIN LOOP FOR ENVIRONMENTS AND PARTITIONS

Require: A PDP system Π of degree (q, m), $T \geq 1$ (time units), $A \geq 1$ (*accuracy* parameter), and an initial configuration C_0.

1: $(p, P) \leftarrow$ *PARTITIONS* (Π) ▷ (Compute the p partitions of rules in the map P)
2: **for** $j \leftarrow 1$ **to** m **do** ▷ (For each environment j)
3: **for** $i \leftarrow 1$ **to** p **do** ▷ (For each partition i)
4: $(\mathcal{T}_{j,i}) \leftarrow$ *INITIALIZATION (j,Π,i,P)* ▷ (The table for environment j and partition i)
5: **end for**
6: **end for**
7: **for** $t \leftarrow 0$ **to** $T - 1$ **do** ▷ (For each transition step)
8: **for** $j \leftarrow 1$ **to** m **do** ▷ (For each environment j)
9: **for** $i \leftarrow 1$ **to** p **do** ▷ (For each partition i)
10: **SELECTION:** ▷ (Selection for environment j and partition i)
11: $B^{j,i}_{sel} \leftarrow \emptyset$, $R^{j,i}_{sel} \leftarrow \emptyset$
12: $(\mathcal{T}^t_{j,i}, C'_t, B^{j,i}_{sel}) \leftarrow$ *PHASE 1* $(j, \Pi, A, C_t, \mathcal{T}_{j,i}, i, P)$
13: $(C'_t, B^{j,i}_{sel}) \leftarrow$ *PHASE 2* $(j, \Pi, C'_t, B^{j,i}_{sel}, \mathcal{T}^t_{j,i}, i, P)$
14: $(R^{j,i}_{sel}) \leftarrow$ *PHASE 3* $(j, \Pi, B^{j,i}_{sel}, i, P)$
15: **end for**
16: **end for**
17: **for** $j \leftarrow 1$ **to** m **do** ▷ (For each environment j)
18: **for** $i \leftarrow 1$ **to** p **do** ▷ (For each partition i)
19: **EXECUTION:** ▷ (Execution for environment j and partition i)
20: $(C_{t+1}) \leftarrow$ *PHASE 4* $(j, \Pi, C'_t, R^{j,i}_{sel}, i, P)$
21: **end for**
22: **end for**
23: **end for**

Preliminary results show an improvement of around 2x of extra speedup when using the partitions to find more parallelism on a K40c GPU with a model of the Bearded Vulture in the Pyrenees [10, 4]. This means that the μ-DCBA implementations usually runs twice faster than the GPU baseline simulator [15].

6 Conclusions and Future Work

PDP systems are a formal framework for ecosystem modelling, whose applications require efficient software simulators. In this concern, GPU-accelerated simulators have been developed so far. However, the designed algorithms for PDP systems, specially DCBA, have several bottlenecks. On the one hand, DCBA assumes that all rules in the system will depend on each other when consuming the left-hand sides. However, this is not always the case, and we can pre-compute partitions of rules actually competing for objects. On the hand, the model designer knows which rules will be applied at each step. An adaptative simulator should be able to use this information to dismiss rules that are known to be non applicable in a certain step.

We have shown how to modify DCBA main loop when implementing these two ideas (adaptative DCBA and DCBA with partitions). These strategies lead to extra speedups (compared with the GPU baseline simulator) of around 2x-2.5x. Let us remark that a simulator implementing these two modes should be used carefully:

- An adaptative simulator of PDP system should be used when deploying a validated model. That is, when the model is already refined and validated by the designer, and the behaviour is already known and proved to work. For example, when using the simulator in virtual experimentation environments.
- A simulator with partitions of PDP systems (e.g. μ-DCBA) can be used not only in virtual experimentation environments, but also in the validation of the model. That is, when the designer is still debugging the model, this strategy can help since it works automatically from the set of rules. However, the performance is not as good as with adaptative simulators (according to our preliminary results), and would require some initial pre-computation for partitions.

Future work includes the development of a μ-DCBA in a stable simulator along with the already existing adaptative simulator. Moreover, we plan to use these improvements to go to the next step and implement parallel parameter calibration methods for the models. We are also working on an automatic inclusion of the GPU simulators inside generic simulation tools such as MeCoSim and P-Lingua, since the only way to use these tools is by a manual protocol [18]. Finally, we are looking into further improvements of the GPU adaptative simulators by including features such as for object counters [2].

Acknowledgements

This work was supported by the research project TIN2017-89842-P (MABICAP), co-financed by *Ministerio de Economía, Industria y Competitividad (MINECO)* of Spain, through the *Agencia Estatal de Investigación (AEI)*, and by *Fondo Europeo de Desarrollo Regional (FEDER)* of the European Union.

References

1. Alhazov, A.: Maximally parallel multiset-rewriting systems: Browsing the configurations. In: Proceedings of the Third Brainstorming Week on Membrane Computing. pp. 1–10. Fénix Editora, Seville, Spain (February 2005), http://www.gcn.us.es/3BWMC/bravolpdf/bravol1.pdf
2. Martínez-del Amor, M.Á., Orellana-Martín, D., Pérez-Hurtado, I., Valencia-Cabrera, L., Riscos-Núñez, A., Pérez-Jiménez, M.J.: Design of specific P systems simulators on GPUs. In: Hinze, T., Rozenberg, G., Salomaa, A., Zandron, C. (eds.) Membrane Computing. pp. 202–207. Springer International Publishing, Cham (2019)
3. Martínez-del Amor, M.A., Pérez-Hurtado, I., García-Quismondo, M., Macías-Ramos, L.F., Valencia-Cabrera, L., Romero-Jiménez, Á., Graciani, C., Riscos-Núñez, A., Colomer, M.A., Pérez-Jiménez, M.J.: DCBA: Simulating Population Dynamics P Systems with Proportional Object Distribution. In: Csuhaj-Varjú, E., Gheorghe, M., Rozenberg, G., Salomaa, A., Vaszil, G. (eds.) Membrane Computing. Lecture Notes in Computer Science, vol. 7762, pp. 257–276. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
4. Cardona, M., Colomer, M.A., Pérez-Jiménez, M.J., Sanuy, D., Margalida, A.: Modeling ecosystems using P systems: the bearded vulture, a case study. In: Corne, D., Frisco, P., Păun, G., Rozenberg, G., Salomaa, A. (eds.) Membrane Computing, Lecture Notes in Computer Science, vol. 5391, pp. 137–156. Springer Berlin Heidelberg (2009)
5. Colomer, M., Pérez-Hurtado, I., Pérez-Jiménez, M., Riscos-Núñez, A.: Comparing simulation algorithms for multienvironment probabilistic P systems over a standard virtual ecosystem. Natural Computing 11(3), 369–379 (2012)
6. Colomer, M.A., Margalida, A., Valencia-Cabrera, L., Palau, A.: Application of a computational model for complex fluvial ecosystems: The population dynamics of zebra mussel Dreissena polymorpha as a case study. Ecological Complexity 20, 116 – 126 (2014)
7. Colomer, M., Margalida, A., Pérez-Jiménez, M.: Population Dynamics P System (PDP) Models: A Standardized Protocol for Describing and Applying Novel Bio-Inspired Computing Tools 8(5), e60698 (2013)
8. Colomer, M., Margalida, A., Sanuy, D., Pérez-Jiménez, M.J.: A bio-inspired computing model as a new tool for modeling ecosystems: The avian scavengers as a case study. Ecological Modelling 222(1), 33–47 (2011)
9. Colomer-Cugat, M.A., García-Quismondo, M., Macías-Ramos, L.F., Martínez-del Amor, M.A., Pérez-Hurtado, I., Pérez-Jiménez, M.J., Riscos-Núñez, A., Valencia-Cabrera, L.: Membrane System-Based Models for Specifying Dynamical Population Systems, pp. 97–132. Springer International Publishing, Cham (2014)
10. Doncel-Ramírez, A.: Simulación Acelerada de Sistemas P de Dinámica de Poblaciones con GPU. Master thesis (Universidad de Sevilla), july 2018
11. García-Quismondo, Manuel and Martínez-del-Amor, M.A., Pérez-Jiménez, M.J.: Probabilistic Guarded P Systems, A New Formal Modelling Framework. In: Gheorghe, M., Rozenberg, G., Salomaa, A., Sosík, P., Zandron, C. (eds.) Membrane Computing. pp. 194–214. Lecture Notes in computer Science, Springer International Publishing, Cham (2014)
12. Martínez-del-Amor, M., Macías-Ramos, L., Valencia-Cabrera, L., Pérez-Jiménez, M.: Parallel simulation of Population Dynamics P systems: updates and roadmap 15(4), 565–573 (2015)

13. Martínez-del-Amor, M., Pérez-Hurtado, I., Pérez-Jiménez, M., Riscos-Núñez, A., Colomer, M.: A new simulation algorithm for multienvironment probabilistic P systems. In: IEEE Fifth International Conference on Bio-Inspired Computing: Theories and Applications (BIC-TA 2010). vol. 1, pp. 59–68 (September 2010)
14. 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: Proceedings of the Tenth Brainstorming Week on Membrane Computing. vol. II, pp. 17–26. Fénix Editora, Seville, Spain (February 2012), http://www.gcn.us.es/10BWMC/10BWMCvolII/papers/parallel-dcba.pdf
15. 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.) Computational Methods in Systems Biology, pp. 247–266. Lecture Notes in Computer Science, Springer Berlin Heidelberg (2012)
16. Martínez-del-Amor, M.A., Pérez-Hurtado, I., Orellana-Martín, D., Pérez-Jiménez, M.J.: Adaptative parallel simulators for bioinspired computing models. Future Generation Computer Systems 107, 469 – 484 (2020)
17. Pérez-Hurtado, I., Orellana-Martín, D., Zhang, G., Pérez-Jiménez, M.J.: P-lingua in two steps: flexibility and efficiency. Journal of Membrane Computing 1(2), 93–102 (Jun 2019)
18. Valencia-Cabrera, L., Martínez-del Amor, M.Á., Pérez-Hurtado, I.: A Simulation Workflow for Membrane Computing: From MeCoSim to PMCGPU Through P-Lingua, pp. 291–303. Springer International Publishing, Cham (2018)
19. Zhang, G., Shang, Z., Verlan, S., Martínez-del-Amor, M.A., Yuan, C., Valencia-Cabrera, L., Pérez-Jiménez, M.J.: An overview of hardware implementation of membrane computing models. ACM Comput. Surv. 53(4) (Aug 2020)