Solving Problems in a Distributed Way in Membrane Computing: dP Systems*

Gheorghe Păun[1,2], Mario J. Pérez-Jiménez[2]

[1] Institute of Mathematics of the Romanian Academy
PO Box 1-764, 014700 Bucureşti, Romania
[2] Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
E-mail: gpaun@us.es, marper@us.es

Summary. Although P systems are distributed parallel computing devices, no explicit way of handling the input in a distributed way in this framework was considered so far. This note proposes a distributed architecture (based on cell-like P systems, with their skin membranes communicating through channels as in tissue-like P systems, according to specified rules of the antiport type), where parts of a problem can be introduced as inputs in various components and then processed in parallel. The respective devices are called dP systems, with the case of accepting strings called dP automata. The communication complexity can be evaluated in various ways: *statically* (counting the communication rules in a dP system which solves a given problem), or *dynamically* (counting the number of communication steps, of communication rules used in a computation, or the number of objects communicated). For each measure, two notions of "parallelizability" can be introduced. Besides (informal) definitions, some illustrations of these idea are provided for dP automata: each regular language is "weakly parallelizable" (i.e., it can be recognized in this framework, using a constant number of communication steps), and there are languages of various types with respect to Chomsky hierarchy which are "efficiently parallelizable" (they are parallelizable and, moreover, are accepted in a faster way by a dP automaton than by a single P automaton). Several suggestions for further research are made.

1 Introduction

P systems are by definition distributed parallel computing devices, [11], [12], [17], and they can solve computationally hard problems in a feasible time, [13], but this efficiency is achieved by a trade-off between space and time, based on the

* The present paper is in press in *Int. J. of Computers, Communication and Control*, Vol. V, No. 2, 2010.

possibility of generating an exponential workspace in a linear time, by means of biologically inspired operations, such as membrane division and membrane creation. However, no class of P systems was proposed where a hard problem can be solved in a distributed parallel way after splitting the problem in parts and introducing these subproblems in components of a P system which can work on these subproblems in parallel and produce the solution to the initial problem by interacting/communicating among each other (like in standard distributed computer science). In particular, no communication complexity, in the sense of [2], [9], [16], was considered for P systems, in spite of the fact that computation (time) complexity is very well developed, [13], and also space complexity was recently investigated, [14]. Some proposals towards a communication complexity of P systems were made in [1], but mainly related to the communication effort in terms of symport/antiport rules used in so-called evolution-communication P systems of [5]. (Note that in communication complexity theory the focus is not on the time efficiency of solving a problem, but the parties involved in the computation just receive portions of the input, in general, distributed in a balanced manner, "as fair as possible" – this distribution introduces an inherent difficulty in handling the input – and then mainly the complexity of the communication needed to parties to handle this input is investigated.)

This note tries to fill in this gap, proposing a rather natural framework for solving problems in a distributed way, using a class of P systems which mixes ingredients already existing in various much investigated types of P systems. Namely, we consider P systems with inputs, in two variants: (i) like in P automata, [6], [10], where a string of symbols is recognized if those symbols are brought into the system from the environment and the computation eventually halts (it is important to note that the string is "read" during the computation, not *before* it), and (ii) in the usual manner of complexity investigations, [13], where an instance of a decision problem is introduced in a P system in the form of a multiset of symbols (this operation takes no time, the computation starts *after* having the code of the problem inside), and the system decides that instance in the end of a computation which sends to the environment one of the special objects `yes` or `no`. Several such systems, no matter of what type, are put together in a complex system which we call *dP system* (from "distributed P system"); the component systems communicate through channels linking their skin membranes, by antiport rules as in tissue-like P systems. When accepting strings by dP systems with P automata as components, the device is called a *dP automaton*.

Such an architecture was already used, with specific ingredients, for instance, in the investigations related to eco-systems, where "local environments" are necessary to be delimited and communication possibilities exist, linking them; details can be found in the recent paper [4].

The way to use a dP system is obvious: a problem Q is split into parts $q_1, q_2, \ldots, q_n$, which are introduced in the n components of the dP system (as in P automata or as in decision P systems), these n systems work separately on their problems, and communicate to each other according to the skin-to-skin rules.

The solution to the problem Q is provided by the whole system (by halting – in the case of accepting strings, by sending out one of the objects `yes` or `no`, etc.). Like in communication complexity, [9], we request the problem to be distributed in a *balanced* way among the components of the dP system, i.e., in "as equal as possible" parts (also an *almost balanced* way to distribute the input among two processors is considered in [9] – no partner takes more than two thirds of the input – which does not seem very natural to be extended to the general case, of n processors).

Several possibilities exist for defining the communication complexity of a computation. We follow here the ideas of [1], and introduce three measures: the number of steps of the computation when a communication rule is used (such a step is called communication step), the number of communication rules used during a computation, and the number of objects transferred among components (by communication rules) during a computation. All these three measures are dynamically defined; we can also consider a static parameter, like in descriptional complexity of Chomsky languages (see a survey in [8]), i.e., the number of communication rules in a dP system.

A problem is said to be "weakly parallelizable" with respect to a given (dynamical) communication complexity measure if it can be split in a balanced way, introduced in the dP system, and solved using a number of communication steps bounded by a constant given in advance; a problem is "efficiently parallelizable" if it is weakly parallelizable and can be solved by a dP system in a more efficient way than by a single P system; more precise definitions are given in the next sections of the paper.

Various possibilities exist, depending on the type of systems (communicating systems, e.g., based on symport/antiport rules, systems with active membranes, catalytic systems, etc.) and the type of problem we consider (accepting strings, decision problems, numerical problems, etc.).

In this note we only sketch the general formal framework and give an illustration, for the case of accepting strings as in P automata. We only show here that all regular languages are weakly parallelizable (only one communication step suffices, hence the weak parallizability holds with respect to all three dynamical measures), and that there are regular, context-free non-regular, context-sensitive non-context-free languages which are efficiently parallelizable with respect to the first two dynamical measures mentioned above (in view of the results in [9], there are linear languages which are not efficiently parallelizable with respect to the number of communicated objects/bits among components).

If we use extended systems (a terminal alphabet of objects is available) and the communication channels among the components of a dP automaton are controlled, e.g., by states, as in [7], or created during the computation, as in [3], then the power of our devices increases considerably: all recursively enumerable languages are weakly parallelizable in this framework.

Many research problems remain to be explored, starting with precise definitions for given classes of P systems, continuing with the study of usefulness

of this strategy for solving computationally hard problems (which problems are weakly/efficiently parallelizable and which is the obtained speed-up for them?), and ending with a communication complexity theory of dP systems, taking into account all measures of complexity mentioned above (for the number of objects communicated among components, which corresponds to the number of bits considered in [9], we can transfer here the general results from communication complexity – note however that in many papers in this area one deals with 2-party protocols, while in our framework we want to have an n-party set-up, and that we are also interested in the time efficiency of the distributed and parallel way of solving a problem).

2 dP Systems – A Preliminary Formalization

The reader is assumed familiar with basics of membrane computing, e.g., from [11], [12], and of formal language theory, e.g., from [15], hence we pass directly to introducing our proposal of a distributed P system. The general idea is captured in the following notion.

A *dP scheme* (of degree $n \geq 1$) is a construct

$$\Delta = (O, \Pi_1, \ldots, \Pi_n, R),$$

where:

1. O is an alphabet of objects;
2. $\Pi_1, \ldots, \Pi_n$ are cell-like P systems with O as the alphabet of objects and the skin membranes labeled with $s_1, \ldots, s_n$, respectively;
3. R is a finite set of rules of the form $(s_i, u/v, s_j)$, where $1 \leq i, j \leq n$, $i \neq j$, and $u, v \in O^*$, with $uv \neq \lambda$; $|uv|$ is called the *weight* of the rule $(s_i, u/v, s_j)$.

The systems $\Pi_1, \ldots, \Pi_n$ are called *components* of the scheme Δ and the rules in R are called *inter-components communication rules*. Each component can take an input, work on it, communicate with other components (by means of rules in R), and provide the answer to the problem in the end of a halting computation. (A delicate issue can appear in the case of components which can send objects to the environment and bring objects from the environment – this happens, for instance, for symport/antiport P systems; in this case we have to decide whether or not the components can exchange objects by means of the environment, or the only permitted communication is done by means of the rules in R. For instance, a "local environment" for each component can be considered, disjoint from the "local environments" of other components, thus preventing the interaction of components by means of other rules than those in R. Actually, the rules in R themselves can be defined between these "local environments" – which is a variant worth to explore. We point out here that also the need of a "local environment" has appeared in the applications of membrane computing to eco-systems investigations, see [4] and its references.)

Now, we can particularize this notion in various ways, depending on the type of systems $\Pi_i, 1 \leq i \leq n$, and the type of problems we want to solve.

For instance, we can define *dP systems with active membranes*, as dP schemes as above, with the components being P systems with active membranes, each of them having a membrane designated as the input membrane. Having a decision problem – consider, e.g., SAT for n variables and m clauses – we can split a given instance of it in parts which are encoded in multisets which are introduced in the components of the dP system. For example, we can introduce the code of each separate clause in a separate component of the dP system. The components start to work, each one deciding its clause, and in the end they communicate to each other the result; if one of the components will find that all m clauses are satisfied, then the whole SAT formula is satisfied. Intuitively, this is a faster way than deciding the formula by means of a single P system with active membranes – but a crucial aspect has been neglected above: in order to say that the formula is satisfied, all the m clauses should be satisfied *by the same truth-assignment*, and this supposes that the m components communicate to each other also which is the assignment which turns true the clauses. That is, besides the usual time complexity of solving the problem we have now to consider the cost of communication among the components and the trade-off between these two parameters should be estimated.

Another interesting case, which will be briefly investigated in the subsequent section, is that of accepting strings in the sense of P automata, [6], [10]; we will come back immediately to this case.

On the other hand, we have several possibilities for estimating "the cost of communication", and we adapt here the ideas from [1].

Let us consider a dP system Δ, and let $\delta : w_0 \Longrightarrow w_1 \Longrightarrow \ldots \Longrightarrow w_h$ be a halting computation in Δ, with w_0 being the initial configuration. Then, for each $i = 0, 1, \ldots, h-1$ we can write:

$ComN(w_i \Longrightarrow w_{i+1}) = 1$ if a communication rule is used in this transition, and 0 otherwise,
$ComR(w_i \Longrightarrow w_{i+1})$ = the number of communication rules used in this transition,
$ComW(w_i \Longrightarrow w_{i+1})$ = the total weight of the communication rules used in this transition.

These parameters can then be extended in the natural way to computations, results of computations, systems, problems/languages. We consider below the case of accepting strings (by $L(\Delta)$ we denote the language of strings accepted by Δ): for $ComX \in \{ComN, ComR, ComW\}$ we define

$ComX(\delta) = \sum_{i=0}^{h-1} ComX(w_i \Longrightarrow w_{i+1})$, for $\delta : w_0 \Longrightarrow w_1 \Longrightarrow \ldots \Longrightarrow w_h$ a halting computation,
$ComX(w, \Delta) = \min\{ComX(\delta) \mid \delta : w_0 \Longrightarrow w_1 \Longrightarrow \ldots \Longrightarrow w_h$ is a computation in Δ which accepts the string $w\}$,
$ComX(\Delta) = \max\{ComX(w, \Delta) \mid w \in L(\Delta)\}$,
$ComX(L) = \min\{ComX(\Delta) \mid L = L(\Delta)\}$.

Similar definitions can be considered for more general decidability problem than accepting strings, then complexity classes can be defined. We do not enter here into details for this general case; in the next section we will briefly consider the specific case of dP automata and of languages.

The previously sketched approach should be investigated in more details. Which is the speed-up for a given problem or class of problems? Clearly, $ComN(\alpha) \leq ComR(\alpha) \leq ComW(\alpha)$, for all valid α. Moreover, in one communication step one can use arbitrarily many communication rules, which therefore move from a component to another one arbitrarily many objects. Anyway, independently of the communication cost, presumably, only a linear speed-up can be obtained by splitting the problem in a given number of parts. Are there problems which however cannot be solved in this framework in a faster way than by using a single P system (with active membranes) provided that the communication cost is bounded (e.g., using communication rules in R only for a constant number of times)? Which is the communication complexity for a given problem or class of problems? Finding suggestive examples can be a first step in approaching such issues.

A case study will be considered in the next section, not for dP systems with active membranes (which, we believe, deserve a separate and detailed examination), but for a distributed version of P automata.

3 dP Automata

We consider now the distributed version of P automata, [6], [10], which are symport/antiport P systems which accept strings: the sequence of objects (because we work with strings and symbol objects, we use interchangeably the terms "object" and "symbol") imported by the system from the environment during a halting computation is the string accepted by that computation (if several objects are brought in the system at the same time, then any permutation of them is considered as a substring of the accepted string; a variant, considered in [6], is to associate a symbol to each multiset and to build a string by such "marks" attached to the imported multisets). The accepted string can be introduced in the system symbol by symbol, in the first steps of the computation (if the string is of length k, then it is introduced in the system in the first k steps of the computation – the P automaton is then called *initial*), or in arbitrary steps. Of course, the initial mode is more restrictive – but we do not enter here into details.

As a kind of mixture of the ideas in [6] and [10] for defining the accepted language, we can consider extended P automata, that is, with a distinguished alphabet of objects, T, whose elements are taken into account when building the accepted string (the other objects taken by the system from the environment are ignored). Here, however, we work with non-extended P automata.

A *dP automaton* is a construct

$$\Delta = (O, E, \Pi_1, \ldots, \Pi_n, R),$$

where $(O, \Pi_1, \ldots, \Pi_n, R)$ is a dP scheme, $E \subseteq O$ (the objects available in arbitrarily many copies in the environment), $\Pi_i = (O, \mu_i, w_{i,1}, \ldots, w_{i,k_i}, E, R_{i,1}, \ldots, R_{i,k_i})$ is a symport/antiport P system of degree k_i (without an output membrane), with the skin membrane labeled with $(i, 1) = s_i$, for all $i = 1, 2, \ldots, n$.

A halting computation with respect to Δ accepts the string $x = x_1 x_2 \ldots x_n$ over O if the components $\Pi_1, \ldots, \Pi_n$, starting from their initial configurations, using the symport/antiport rules as well as the inter-components communication rules, in the non-deterministically maximally parallel way, bring from the environment the substrings $x_1, \ldots, x_n$, respectively, and eventually halts.

The dP automaton is synchronized, a universal clock exists for all components, marking the time in the same way for the whole dP automaton. It is also important to note that we work here in the non-extended case, all input symbols are recorded in the string. In this way, at most context-sensitive languages can be recognized.

The three complexity measures $ComN, ComR, ComW$ defined in the previous section can be directly introduced for dP automata (and they were formulated above for this case). With respect to them, we can consider two levels of parallelizability.

A language $L \subseteq V^*$ is said to be (n, m)-*weakly ComX parallelizable*, for some $n \geq 2, m \geq 1$, and $X \in \{N, R, W\}$, if there is a dP automaton Δ with n components and there is a finite subset F_Δ of L such that each string $x \in L - F_\Delta$ can be written as $x = x_1 x_2 \ldots x_n$, with $||x_i| - |x_j|| \leq 1$ for all $1 \leq i, j \leq n$, each component Π_i of Δ takes as input the string $x_i, 1 \leq i \leq n$, and the string x is accepted by Δ by a halting computation δ such that $ComX(\delta) \leq m$. A language L is said to be *weakly ComX parallelizable* if it is (n, m)-weakly $ComX$ parallelizable for some $n \geq 2, m \geq 1$.

Two conditions are here important: (i) the string is distributed in equal parts, modulo one symbol, to the components of the dP automaton, and (ii) the communication complexity, in the sense of measure $ComX$, is bounded by the constant m.

We have said nothing before about the length of the computation. That is why we also introduce a stronger version of parallelizability.

A language $L \subseteq V^*$ is said to be (n, m, k)-*efficiently ComX parallelizable*, for some $n \geq 2, m \geq 1, k \geq 2$, and $X \in \{N, R, W\}$, if it is (n, m) weakly ComX parallelizable, and there is a dP automaton Δ such that

$$\lim_{x \in L, |x| \to \infty} \frac{time_\Pi(x)}{time_\Delta(x)} \geq k,$$

for all P automata Π such that $L = L(\Pi)$ ($time_\Gamma(x)$ denotes here the smallest number of steps needed for the device Γ to accept the string x). A language L is said to be *efficiently ComX parallelizable* if it is (n, m, k)-efficiently $ComX$ parallelizable for some $n \geq 2, m \geq 1$, $k \geq 2$.

Note that in the case of dP automata, the duration of a computation may also depend on the way the string is split in substrings and introduced in the

components of the system; in a natural way, one of the most efficient distribution of the string and shortest computation are chosen. Of course, as larger the constant k as better.

Moreover, while $time_{\Delta}(x)$ is just given by means of a construction of a suitable dP automaton Δ, $time_{\Pi}(x)$ should be estimated *with respect to all P automata Π*.

An example is worth considering in order to illustrate this definition. Let us examine the dP system from Figure 1 – the alphabet of objects is $O = \{a, b, c, d, c_1, c_2, \#\}$, and $E = \{a, b\}$.

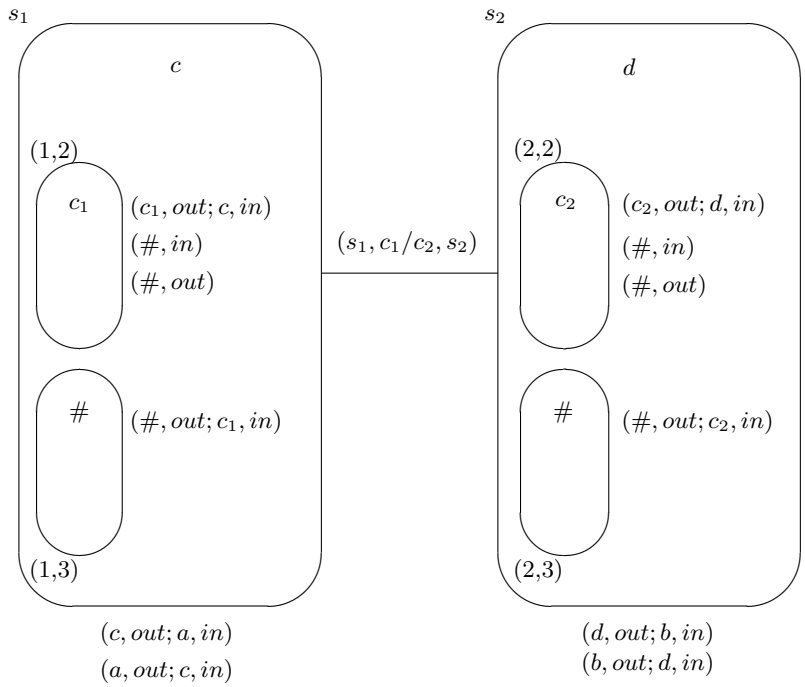

Fig. 1. An example of a dP automaton

Clearly, component Π_1 (in the left hand side of the figure) can only bring objects a, c inside, and component Π_2 (in the right hand side of the figure) can only bring objects b, d inside. In each step, only one of a, c, alternately, enters Π_1 and only one of b, d, alternately, enters Π_2 (note that we do not need objects c, d to be present initially in the environment, while one copy of each a and b is sufficient). The computation of each component can stop only by "hiding" the "carrier objects" c, d inside an inner membrane, and this means releasing c_1 in Π_1 and c_2 in Π_2. If these objects are not released at the same time in the two components, so that the exchange rule $(s_1, c_1/c_2, s_2)$ can be used, then, because of

the maximal parallelism, the object c_1 should enter membrane (1,3), and object c_2 should enter membrane (2,3); in each case, the trap-object # is released, and the computation never stops: the object # oscillates forever across membrane (1,2) in Π_1 and across membrane (2,2) in Π_2.

Consequently, the two strings accepted by the two components of Δ should have the same length, that is the language accepted by the system is

$$L(\Delta) = \{(ac)^s(bd)^s \mid s \geq 0\}.$$

Note the crucial role played here by the fact that the system is synchronized, and that a computation which accepts a string $x_s = (ac)^s(bd)^s$, hence of length $4s$, lasts $2s + 2$ steps ($2s$ steps for bringing objects inside, one step when objects c, d are introduced in an inner membrane, and one inter-components communication step), with one of these steps being a communication between components.

Obviously, if we recognize a string $x_s = (ac)^s(bd)^s$ as above by means of a usual symport/antiport P system, then, because no two symbols of the string can be interchanged, no two adjacent symbols can be introduced in the system at the same step, hence the computation lasts at least as many steps as the length of the string, that is, $4s$. This shows that our language is not only $(2, r)$-weakly $ComX$ parallelizable, but also $(2, r, 2)$-efficiently $ComX$ parallelizable, for $(r, X) \in \{(1, N), (1, R), (2, W)\}$.

This conclusion is worth formulating as a theorem.

Theorem 1. *The language $L = \{(ac)^s(bd)^s \mid s \geq 0\}$ is efficiently $ComX$ parallelizable, for all $X \in \{N, R, W\}$.*

Note that this language is not regular (but it is linear, hence also context-free).

The previous construction can be extended to dP automata with three components: Π_1 inputs the string $(ac)^s$, Π_2 inputs $(bd)^s$, and Π_3 inputs $(ac)^s$, then Π_1 produces the object c_1, Π_2 produces two copies of c_2, and Π_3 produces the object c_3. Now, c_1 is exchanged for one copy of c_2 from Π_2 and c_3 for the other copy, otherwise the computation never stops. The recognized language is $\{(ac)^s(bd)^s(ac)^s \mid s \geq 0\}$.

This language is not context-free, hence we have:

Theorem 2. *There are context-sensitive non-context-free languages which are efficiently $ComX$ parallelizable, for all $X \in \{N, R, W\}$.*

The previous two theorems show that the distribution, in the form of dP systems, is useful from the time complexity point of view, although only one communication step is performed and only one communication rule is used at that step. Moreover, the proofs of the two theorems show that, in general, languages consisting of strings with two well related halves (but not containing "too much" information in each half of the string, besides the length), are weakly parallelizable, and, if no two adjacent symbols of the strings can be interchanged, then these languages are efficiently parallelizable.

We have said nothing above about regular languages – this is the subject of the next section.

4 All Regular Languages are Weakly Parallelizable

The assertion in the title of this section corresponds to Theorem 2.3.5.1 in [9], which states that for each regular language there is a constant k which bounds its (2-party) communication complexity. The version of this result in terms of weak $ComX$ parallelizability is shown by the following construction. Consider a non-deterministic finite automaton $A = (Q, T, q_0, F, P)$ (set of states, alphabet, initial state, final states, set of transition rules, written in the form $qa \to q'$, for $q, q' \in Q, a \in T$). Without any loss of generality, we may assume that all states of Q are reachable from the initial state (for each $q \in Q$ there is $x \in T^*$ such that $q_0 x \Longrightarrow^* q$ with respect to transition rules in P). We construct the following dP automaton:

$$
\begin{aligned}
\Delta &= (O, E, \Pi_1, \Pi_2, R), \text{ where :}\\
O &= Q \cup T \cup \{d\}\\
&\quad \cup \{(q, q') \mid q, q' \in Q\}\\
&\quad \cup \{\langle q, q_f\rangle \mid q \in Q, q_f \in F\}\\
&\quad \cup \{\langle q\rangle \mid q \in Q\},\\
E &= O - \{d\},\\
\Pi_1 &= (O, [_{s_1} [_{1,2} \ ]_{1,2}]_{s_1}, q_0, \lambda, E, R_{s_1}, R_{1,2}),\\
R_{s_1} &= \{(q, out; q'a, in) \mid qa \to q' \in P\}\\
&\quad \cup \{(q, out; \langle q'\rangle a, in) \mid qa \to q' \in P\},\\
R_{1,2} &= \{(\langle q\rangle, in),\ (\langle q\rangle, out) \mid q \in Q\},
\end{aligned}
$$

$$
\begin{aligned}
\Pi_2 &= (O, [_{s_2} \ ]_{s_2}, d, E, R_{s_2}),\\
R_{s_2} &= \{(d, out; (q, q')a, in) \mid qa \to q' \in P, q \in Q\}\\
&\quad \cup \{((q, q'), out; (q, q'')a, in) \mid q'a \to q'' \in P, q \in Q\}\\
&\quad \cup \{((q, q'), out; \langle q, q_f\rangle a, in) \mid q'a \to q_f \in P, q \in Q, q_f \in F\},\\
R &= \{(s_1, \langle q\rangle / \langle q, q_f\rangle, s_2) \mid q \in Q, q_f \in F\}.
\end{aligned}
$$

The first component analyzes a prefix of a string in $L(A)$, the second component analyzes a suffix of a string in $L(A)$, first guessing a state $q \in Q$ from which the automaton starts its work. At some moment, the first component stops bringing objects inside by taking from the environment a symbol $\langle q'\rangle$ for some $q' \in Q$, reached after parsing the prefix of the string in $L(A)$. This object will pass repeatedly across the inner membrane of Π_1. The second component can stop if a state q' is reached in the automaton A for which no rule $q'a \to q''$ exists in P (and then Δ never stops, because its first component never stops), or after reaching a state in F, hence introducing an object of the form $\langle q, q_f\rangle$ for some $q_f \in F$. Note that q is the state chosen initially and always stored in the first position of objects (q_1, q_2) used by Π_2. The computation can halt only by using a communication

rule from R, and this is possible only if $q = q'$ – the first component has reached the state of A which was the state from which the second component started its work. Consequently, the concatenation of the two strings introduced in the system by the two components is a string from $L(A)$. Thus, the language $L(A)$ is weakly parallelizable.

Now, consider a regular language such that no two adjacent symbols in a string can be permuted (take an arbitrary regular language L over an alphabet V and a morphism $h : V^* \longrightarrow (V \cup \{c\})^*$, where c is a symbol not in V, such that $h(a) = ac$ for each $a \in V$). Then, clearly, if the two strings accepted by the two components of the dP automaton Δ are of equal length (note that the strings of $h(L)$ are of an even length), then the time needed to Δ to accept the whole string is (about) half of the time needed to any P automaton Π which accepts the same language. This proves that the language $h(L)$ is efficiently parallelizable, hence we can state:

Theorem 3. *Each regular language is weakly ComX parallelizable, and there are efficiently ComX parallelizable regular languages, for all* $X \in \{N, R, W\}$.

Of course, faster dP automata can be constructed, if we use more than two components. However, it is not clear whether dP automata with $n+1$ components are always faster than dP automata with n components – this might depend on the structure of the considered language (remember that the distribution of the input string to the components of the dP automaton must be balanced). More specifically, we expect that there are (n, m) weakly parallelizable languages which are not, e.g., $(n + 1, m)$ weakly parallelizable; similar results are expected for efficiently parallelizable languages.

A natural question is how much the result in Theorem 3 can be extended. For instance, is a similar result true for the linear languages, or for bigger families of languages? According to Theorem 2.3.5.4 in [9], this is not true for measures $ComR$ and $ComW$, the recognition of context-free languages (actually, the language L_R at page 78 of [9] is linear) have already the highest communication complexity (in 2-party protocols), a linear one with respect to the length of the string. Thus, the number of communication rules used by a dP automaton during a computation cannot be bounded by a constant. The case of measure $ComN$ remains to be settled: is it possible to have computations with a bounded number of communication steps, but with these steps using an unbounded number of rules? We conjecture that even in this case, languages of the form $\{x\ mi(x) \mid x \in \{a, b\}^*\}$, where $mi(x)$ is the mirror image of x (such a language is minimally linear, i.e., can be generated by a linear grammar with only one nonterminal), are not weakly $ComN$ parallelizable.

Many other questions can be raised in this framework. For instance, we can consider families of languages: (n, m)-weakly $ComX$ parallelizable, weakly $ComX$ parallelizable, (n, m, k)-efficiently $ComX$ parallelizable, and efficiently $ComX$ parallelizable. Which are their properties: interrelationships and relationships with families in Chomsky hierarchy, closure and decidability properties, hierarchies on various parameters, characterizations and representations, etc.

Then, there is another possibility of interest, suggested already above: the static complexity measure defined as the cardinality of R, the set of communication rules. There is a substantial theory of descriptional complexity of (mainly context-free) grammars and languages, see [8], which suggests a lot of research questions starting from $ComS(\Delta) = card(R)$ (with "S" coming from "static") and extended to languages in the natural way ($ComS(L) = \min\{ComS(\Delta) \mid L = L(\Delta)\}$): hierarchies, decidability of various problems, the effect of operations with languages on their complexity, etc.

5 The Power of Controlling the Communication

In the previous sections the communication rules were used as any rule of the system, non-deterministically choosing the rules to be applied, with the communication rules competing for objects with the inner rules of the components, and observing the restriction of maximal parallelism. However, we can distinguish the two types of rules, "internal evolution rules" (transition rules, symport/antiport rules, rules with active membranes, etc.) and communication rules. Then, as in [1], we can apply the rules according to a priority relation, with priority for evolution rules, or with priority for communication rules. Moreover, we can place various types of controls on the communication channel itself. For instance, because the communication rules are antiport rules, we can associate with them promoters or inhibitors, as used in many places in membrane computing.

A still more natural regulation mechanism is to associate *states* with the channels, like in [7]. In this case, the communication rules associated with a pair (i, j) of components Π_i, Π_j are of the form $(q, u/v, q')$, where q, q' are elements of a given finite set Q of states; initially, the channel is assumed in a given state q_0. A rule as above is applied only if the cannel is in state q – and the antiport rule $(i, u/v, j)$ can be applied; after exchanging the multisets u, v among the two components Π_i, Π_j, the state of the channel is changed to q'.

An important decision should be made in what concerns the parallelism. In [7], the channel rules are used in the sequential mode, but we can also consider two types of parallelism: (i) choose a rule and use it as many times as made possible by the objects in the two components, or (ii) apply at the same time all rules of the form $(q, u/v, q')$ for various u and v (but with the same q and q'), in the non-deterministic maximally parallel way. In the result discussed below, any of these two possibilities works – and the result is somewhat surprising:

Theorem 4. *Any recursively enumerable language L is $(2, 2)$-weakly ComN and ComR parallelizable and has $ComS(L) \leq 2$, with respect to extended dP automata with channel states.*

We do not formally prove this assertion, but we only describe the (rather complex, if we cover all details) construction of the suitable dP automaton.

Take a recursively enumerable language $L \subseteq T^+$, for some $T = \{a_1, a_2, \ldots, a_n\}$. For each string $w \in T^+$, let $val_{n+1}(w)$ be the value of w when considered as a number in base $n+1$, using the digits $a_1, a_2, \ldots, a_n$ interpreted as $1, 2, \ldots, n$, without also using the digit zero. We extend the notation to languages, in the natural way: $val_{n+1}(L) = \{val_{n+1}(w) \mid w \in L\}$. Clearly, L is recursively enumerable if and only if $val_{n+1}(L)$ is recursively enumerable, and the passage from strings w to numbers $val_{n+1}(w)$ can be done in terms of P automata (extended symport/antiport P systems are universal, hence they can simulate any Turing machine; this is one of the places where we need to work with extended systems, as we need copies of a and b – see below – to express the values of strings, and such symbols should be taken from the environment without being included in the accepted strings).

Construct now a dP automaton Δ with two components, Π_1 and Π_2, working as follows. The component Π_1 receives as input a string $w_1 \in T^*$ and Π_2 receives as input a string $w_2 \in T^*$, such that $w_1 w_2$ should be checked whether or not it belongs to the language L. Without loss of generality, we may assume that $|w_1| \in \{|w_2|, |w_2|+1\}$ (we can choose a balanced distribution of the two halves of the string). In the beginning, the state of the channel between the two components is q_0.

Both components start to receive the input symbols, one in each time unit; the component Π_1 transforms the strings w_1 in $val_{n+1}(w_1)$ copies of a symbol a, and Π_2 transforms the string w_2 in $val_{n+1}(w_2)$ copies of a symbol b. When this computation is completed in Π_1, a special symbol, t, is introduced. For this symbol, we provide the communication rule $(q_0, t/\lambda, q_1)$, whose role is to change the state of the channel. We also consider the rule $(q_1, a/\lambda, q_2)$. Using it in the maximally parallel way, all symbols a from Π_1 are moved to Π_2, in one communication step.

Because we have considered w_1 at least of the length of w_2 and we also need two steps for "opening" the channel and for moving the symbols a across it, we are sure that in this moment in Π_2 we have, besides the $val_{n+1}(w_1)$ copies of a, $val_{n+1}(w_2)$ copies of b. The second component takes now these copies of a and b and computes $val_{n+1}(w_1 w_2)$, for instance, as the number of copies of an object c. After that, Π_2 checks whether or not $val_{n+1}(w_1 w_2) \in val_{n+1}(L)$. If the computation halts, then the string $w_1 w_2$ is accepted, it belongs to the language L.

Note that the dP automaton Δ contains two communication rules (hence $ComS(L) \leq 2$) and that each computation contains two communication steps (hence $ComN(L) \leq 2$), in each step only one rule being used (hence $ComR(L) \leq 2$). These observations complete the proof of the theorem.

Of course, $ComW(\Delta) = \infty$. (Similarly, if we define $ComR$ taking into account the multiplicity of using the rules, then also $ComR$ can be considered infinite – hence the assertion in the theorem remains to be stated only for the measure $ComN$.)

Instead of changing channel states as above, we can assume that the channel itself switches from "virtual" to "actual", like in population P systems, [3]: the channel is created by object t produced by Π_1, and then used for moving a from

Π_1 to Π_2 by a usual communication rule (which, by definition, is used in the maximally parallel way).

Anyway, the conclusion of this discussion is that the results we obtain crucially depend on the ingredients we use when building our dP systems (as well as on the chosen definitions for complexity measures and types of parallelizability).

6 Closing Remarks

The paper proposes a rather natural way (using existing ingredients in membrane computing, bringing no new, on purpose invented, stuff into the stage) for solving problems in a "standard" distributed manner (i.e., splitting problems in parts, introducing them in various component "computers", and constructing the solution through the cooperation of these components) in the framework of membrane computing. So called dP schemes/systems were defined, and two notions of parallelizability were proposed and briefly investigated for the case of dP automata (accepting strings).

A lot of problems and research topics were suggested. The reader can imagine also further problems, for instance, transferring in this area notions and questions from the communication complexity theory, [9], considering other types of P systems (what about spiking neural P systems, where we have only one type of objects and no antiport-like rules for communicating among components?), maybe using unsynchronized P systems, non-linear balanced input, and so on and so forth. We are convinced that dP systems are worth investigating.

Note. During the recent Brainstorming Week on Membrane Computing, 1-5 of February 2010, Sevilla, Spain, several comments about the definitions and the results of this paper were made, especially by Erzsébet Csuhaj-Varú, György Vaszil, Rudolf Freund, and Marian Kögler. Several continuations of this paper are now in preparation; the interested reader is requested to check the bibliography from [17], in particular, the Brainstorming proceedings volume.

Acknowledgements

This work is supported by Proyecto de Excelencia con Investigador de Reconocida Valía, de la Junta de Andalucía, grant P08 – TIC 04200. Useful discussions with Henry Adorna are gratefully acknowledged.

References

1. H. Adorna, Gh. Păun, M.J. Pérez-Jiménez: On communication complexity in evolution-communication P systems. Manuscript, 2009.
2. L. Babai, P. Frankl, J. Simon: Complexity classes in communication complexity. *Proc. 27th Annual Symp. Founf. Computer Sci.*, 1986, 337–347.

3. F. Bernardini, M. Gheorghe: Population P systems. *J. Universal Computer Sci.*, 10, 5 (2004), 509–539.
4. M. Cardona, M.A. Colomer, A. Margalida, I. Pérez-Hurtado, M.J. Pérez-Jiménez, D. Sanuy: A P system based model of an ecosystem of some scavenger birds. *Membrane Computing. Proc. WMC10, Curtea de Argeş, 2009* (Gh. Păun et al., eds.), LNCS 5957, Springer, 2010, 182–195.
5. M. Cavaliere: Evolution-communication P systems. *Membrane Computing. Proc. WMC 2002, Curtea de Argeş* (Gh. Păun et al., eds.), LNCS 2597, Springer, Berlin, 2003, 134–145.
6. E. Csuhaj-Varjú: P automata. *Membrane Computing. Proc. WMC5, Milano, 2004* (G. Mauri et al., eds.), LNCS 3365, Springer, Berlin, 2005, 19–35.
7. R. Freund, Gh. Păun, M.J. Pérez-Jiménez: Tissue-like P systems with channel-states. *Theoretical Computer Sci.*, 330, 1 (2005), 101–116.
8. J. Gruska: Descriptional complexity of context-free languages. *Proc. Symp. on Mathematical Foundations of Computer Science, MFCS*, High Tatras, 1973, 71–83.
9. J. Hromkovic: *Communication Complexity and Parallel Computing: The Application of Communication Complexity in Parallel Computing*. Springer, Berlin, 1997.
10. M. Oswald: *P Automata*. PhD Thesis, TU Vienna, 2003.
11. Gh. Păun: *Membrane Computing. An Introduction*. Springer, Berlin, 2002.
12. Gh. Păun, G. Rozenberg, A. Salomaa, eds.: *Handbook of Membrane Computing*. Oxford University Press, 2010.
13. M.J. Pérez-Jiménez: A computational complexity theory in membrane computing. *Membrane Computing. Proc. WMC10, Curtea de Argeş, 2009* (Gh. Păun et al., eds.), LNCS 5957, Springer, 2010, 125–148.
14. A.E. Porreca, A. Leporati, G. Mauri, C. Zandron: Introducing a space complexity measure for P systems. *Intern. J. Computers, Communications and Control*, 4, 3 (2009), 301–310.
15. G. Rozenberg, A. Salomaa, eds.: *Handbook of Formal Languages*. 3 volumes, Springer, Berlin, 1998.
16. A.C. Yao: Some complexity questions related to distributed computing. *ACM Symposium on Theory of Computing*, 1979, 209–213.
17. The P Systems Website: `http://ppage.psystems.eu`.