Scalable Computing: Practice and Experience
Volume 7, Number 3, pp. 39–49. http://www.scpe.org
ISSN 1895-1767

A PARALLEL RULE-BASED SYSTEM AND ITS EXPERIMENTAL USAGE IN MEMBRANE COMPUTING *

DANA PETCU†

Abstract. Distributed or parallel rule-based systems are currently needed for real applications. The proposed architecture of such a system is based on a wrapper allowing the cooperation between several instances of the rule-based system running on different computers of a cluster.

As case study a parallel version of the Java Expert System Shell is built. Initial tests show its efficiency when running classical benchmarks. Moreover, this parallel version of Jess is successfully used to accelerate current simulators for membrane computing.

Key words. Rule-based system, cluster computing, membrane computing

1. Introduction. The suites of benchmarks, like the one reported in [30], show that the rule-based or production systems, running on the current hardware, need hours to give the solutions when the number of rules to be fired are measured in thousands. In this context, parallel, distributed or grid version of those systems are welcome. The strategies to improve the speedup using multiple processors were already discussed in the last twenty years. Details are given in Section 2.

The strategy adopted here to build parallel rule-based systems has a high degree of flexibility: to construct a wrapper for the parallel system which allows cooperation between its sequential instances. The wrapper must be the same or slightly different for most of the available production or rule-based systems.

Jess, Java Expert System Shell [23], has been taken as an example for this study mainly due to its openness towards the computing environment (communication via sockets, ability to create Java objects and call Java methods). It is a rule-based programming environment written in Java. Inspired by Clips expert system shell [13], it has grown into a complete environment of its own, being today one of the fastest rule engines available. Jess uses the Rete algorithm [19] to compile rules, an efficient mechanism for solving the difficult many-to-many matching problem. With Jess one can build software that has the capacity to reason using knowledge supplied in the form of declarative rules. Jess is used in agent frameworks, expert systems from medicine to cryptography, intelligent tutoring, robotics, and so on [17].

The structure and functionality of Parallel Jess are detailed in Section 3. Some preliminary tests on Parallel Jess were performed showing its efficiency when using a cluster of workstations. A report on these experiments is presented in Section 4.

Our motivation to build a parallel version of a rule-based system comes from a practical request: the need of a faster rule-based simulator for membrane computing. Membrane computing (or P-systems) is a domain that was qualified as emergent research front in computer science [7]. The class of P systems promise polynomial time solutions to NP-complete problems, the clue being the ability to generate an exponential working space in a linear time by means of bio-inspired operations such as cell division, membrane creation, or content replication.

A P system is a distributed parallel computing model inspired from the way the living cells process chemical compounds, energy, and information. P systems are inherently parallel and, in many variants, they also exhibit an intrinsic non-determinism, hard to be caught by sequential computers. The web page [38] contains an extensive bibliography of hundreds of papers devoted to P systems.

There are several attempts to simulate P systems on existing computers. They have both didactic and scientific values. Details can be found in Section 5.

We started from an existing Clips simulator described in [32] which allows the study of the evolution of P systems with active membranes based on production system techniques. The set of rules and the configurations in each step of the evolution are expressed as facts in a knowledge base. In Section 6 we first prove that a splitting technique of the membranes in several Java threads running embedded Jess can lead to faster simulation. Then we use the Parallel Jess to further speedup the simulation.

Further directions of concept development and some conclusions can be found in Section 7.

*This work was partially supported by the projects CEEX-I03-47-2005-ForMOL and CNCSIS-949-2004-CompGrid funded by Romanian Ministry of Research.

†Computer Science Department, Western University of Timişoara, and Institute e-Austria Timişoara, B-dul Vasile Pârvan 4, 300223 Timişoara, Romania (petcu@info.uvt.ro).

2. Parallel production or rule-based systems. A production system (PS) consists in a working memory, a set of rules and an inference engine. The working memory is a global database of data (elements) representing the system state. A rule is a condition-action pair. The inference engine is based on a three-phase cyclic execution model of condition evaluations (matching), conflict-resolution and action firing. An instantiation is a rule with a set of working memory elements. A conflict set is the set of all instantiations. Firing an instantiation can add, delete or modify elements in the working memory.

In a sequential environment, conflict-resolution selects one instantiation from the conflict set for firing. In a parallel environment, multiple instantiations can be selected to fire simultaneously.

Encompassing a high degree of parallelism, the time performance of production systems can be improved through parallel processing. The interest in parallel production systems has been raised after 1984 and the first parallel implementations were already available at the beginning of the last decade. Amaral in [1] presents a comprehensive synthesis of the efforts made before 1994 in providing parallel firing systems. We mention here only the most representative contributions: PESA—a parallel architecture for PSs; Rubic—multiprocessor for PSs, multiple rule firing PSs on hypercube; PARULEL—parallel rule processing using meta-rules; PPL—parallel production language.

Concerning the Jess predecessor, Clips, there are several parallel and distributed implementations produced mainly before 1994: for parallel shared memory architectures [8] and [40]; for hypercubes [22, 29]; for parallel distributed memory architectures—dClips [25] and PClips [26]; for distributed system architectures, [20] and using PVM [27]. For example, in the version reported in [22] and developed in 1994 to run on Intel hypercubes new added commands allows parallel calls. A complete version of Clips runs on each node of the hypercube, only rule-level parallelism is supported, and parallel commands enable the assertion and retraction of facts to and from remote nodes working memory.

Unfortunately, none of the above mentioned parallel and distributed versions of rule-based systems are anymore available in the public domain.

The interest in distributed rule-based engines has appear again in last five years in connection with Jess. Recently Jess was used in conjunction with a Java implementation of an actor model [16] to write distributed artificial intelligence applications. In the computational environment each Jess is an active independent computational entity with the ability to communicate with other Jess instances. The OKEANOS middleware [41] provides an infrastructure for mobile agents that access computational services and communicate by passing messages. The agents are implemented in Java and contain rule-based knowledge interpreted by Jess. They consist of two parts: one is responsible for managing messages and for transforming them into Jess-based declarative rules, and the other one, the Jess engine, interprets the incoming rules, new facts are added to the knowledge base, or existing facts are retracted from it, depending on the content of the incoming messages. The resulting knowledge base of a Jess-agent determines its state.

The parallel matching approach parallelizing only the match phase leads to a limited speedup by the sequential execution of rules [43]. The multiple rule firing approach parallelizing the match phase and the act phase by firing multiple rules in parallel is more promising, but supplementary costs are due to synchronization needs. Special techniques like copy-and-constraints, compatible rules, analysis of data dependency graph have been used with success to increase the parallelism. Those techniques are general and do not exploit the parallelism specific to the application domains, as it is the case here. The task-level parallelism approach based on the functional decomposition of the problem into a hierarchy of tasks can lead to better results that the above mentioned ones, but the techniques tend to be ad hoc [43].

In this context we are interested to build a parallel distributed memory version of Jess based on task parallelism.

3. Parallel Jess. In our tests, Jess was chosen over several other rule-based systems because of its active development and support, tight interaction with Java programs, and expressiveness. The Jess rule language includes elements not present in many other production systems, such as arbitrary combinations of boolean conjunctions and disjunctions. Its scripting language is powerful enough to generate full applications entirely within the Jess system [17]. The core Jess language is still compatible with Clips, in that many Jess scripts are valid Clips scripts and vice-versa. Jess adds many features to Clips, including backwards chaining, working memory queries, and the ability to manipulate and directly reason about Java objects. Jess is also a powerful Java scripting environment, from which one can create Java objects and call Java methods without compiling any Java code.

As stated above, we are interested in building a parallel version of Jess based on task parallelism. The target architecture that we consider is a homogeneous cluster of workstations. Extensions towards Grid architecture are not excluded.

Building a rule-based application based on socket communication facilities can distract the user from its main aim, to solve a concrete problem. To avoid this, a middleware is needed.

For the Parallel Jess structure we adopted a modular scheme composed by Instances, Connectors, and Messengers. The Instances are the Jess kernels.

The modular approach has been taken in consideration to simplify the task of migration towards other production systems or other message passing protocols. In the case of a such migration, the Connector must be rewritten to adopt the new production system, or the Messenger must be rewritten to adopt the new message passing protocol. A graphical representation of the proposed structure is given in [35].

The task farming model is used. Each Jess instance has a unique identifier stored in a Jess variable p. The are two types of Jess Instances: the one controlled by the user normally using a local Jess interface—the master or farmer, labeled 0 –, and the ones controlled by the user via the system—the slaves or workers, labeled starting from 1. While the master's Jess runs in the interactive version, a worker's Jess is running in a Java embedded version. The first one is launched by the user. The later one is possible to run on a remote computing platform and it is launched by a Java code.

The first part of the Jess wrapper for the parallel version, the Connector, is written in Java. Each Jess instance has one corresponding Connector. The Jess instance acts as a client and contacts via socket its Connector, the server. The Connector uses the standard Java ServerSockets methods. If the connection is established, the Connector interprets the Jess special incoming requests. Those requests are concerning either a communication or an action on other Jess instance: send or receive an information, launch or kill other instances. A message in transit is a string containing a command written in Jess language.

The second part of the Jess wrapper is the Messenger. Each Messenger is associated with one Connector and its purpose is to execute the commands received by the Connector, and to communicate with the Messengers associated with the other Connector and Jess instances. The Messenger is written in Java and JPVM [24], a Java implementation of Parallel Virtual Machine.

JPVM was selected instead mpiJava or other similar environments for message passing, due to its ability to dynamically create and destroy tasks. Adopting a PVM variant, the user is absolved of the duties to nominate the hosts on which the Jess instances are running, to treat sequentially the incoming messages, or to check the status of the machines on which the Jess instances are running. Asynchronous incoming message delivery, checks for incoming high priority messages, or hierarchies of Jess instances are possible.

The set of new commands added to Jess language is the minimal one required to implement a message passing interface. These commands are described in Table 3.1.

A very simple example of using Parallel Jess is provided in Figure 3.1. The commands given by the user to the Jess interface are displayed. JPVM daemon must be already activated. First, the user loads the new Jess function definitions by specifying the ParJess file. The local Connector & Messenger are started, and the connection between the Jess instance and the Connector is establish by calling the connection function. Then two new Jess instances are launched. Depending on the current JPVM configuration those Jess instances

TABLE 3.1
Jess functions defined in ParJess.clp

Function	Description
connection	Establishes the socket connection between the Jess instance and the Connector
kernels n	Launches n embedded Jess instances
kill n	Stops the nth embedded Jess instance
send n t s	Sends to the nth Jess instance the message labeled t and containing the string s (non-blocking function); if n is -1, the message is broadcast to all Jess instances
recv n t	Returns a string representing a message labeled t and received from the nth Jess kernel; if n is -1, it is accepted the first message from any kernel; if t is -1, it is accepted the message with any label; if no message has arrive, the execution is blocked until the message arrives
prob n t	Tests if a message labeled t send by the nth instance has arrive (non-blocking function); returns "t" or "f"; the meaning of -1 is the same as in the case of recv
stop	Stops all embedded Jess instances, closes the Connector-to-Messenger socket connection

```
                    Jess> (batch ‘‘ParJess.clp")
                                TRUE
                    Jess> (connection)
                                Connection established
                    Jess> (kernels 2)
                                2 kernels launched
                    Jess> (send -1 1 ‘‘(sqrt (+ ?*p* 1))")
                                Multicast successful
                    Jess> (recv 1 2)
‘‘1.4142135623730951"
                    Jess> (recv 2 2)
‘‘1.7320508075688772"
                    Jess> (stop)
                                Connection closed
```

FIG. 3.1. *A simple example*

can run on the same machine or on remote machines. A simple command to perform $\sqrt{p+1}$ where p is the instance identifier is send to the new Jess instances. The first number in the send command indicates the destination instance; -1 is used to broadcast the request to all instances. The label 1 in the send command express the fact that it is the first request addressed to the destination instance. The results from each instance are received in a sequence of three commands. The label 2 used in each receive command means that is the second action concerning the specified instance. The results are received in the form of strings. Finally the three Jess instances (one master and two slaves) are killed, the local socket connection is closed, and all the Connectors and Messengers are shutdown.

At start, each embedded Jess instance receives, from the master instance, its identifier and the list of the identifiers of the other instances in order to be able to communicate with them. The file ParJess is loaded implicitly and the workers enter in an infinite loop in which they wait to receive and execute commands from the Jess master and the other workers. The master messages have higher priority compared to the worker ones.

4. Benchmarks. To measure the efficiency of Parallel Jess several tests are needed. We used the classical benchmark from [30].

The particular test problem presented here is the Miss Manners problem. It is the problem of finding an acceptable seating arrangement for guests at a dinner party, by attempting to match people with the same hobbies, and to seat everyone next to a member of the opposite sex. The classical solution employs a depth-first search approach to the problem. The variables of the problem are the number of guests and chairs, the maximum and minimum numbers of hobbies (e.g. 128 guests, 128 chairs, max 3 hobbies, min 2 hobbies). The computation stops when a solution is found or there is no solution.

A Jess version of the solution can be found at [18]. The data are generated randomly, the number of guests of opposite sex being equal. For each guest, name, sex and list of hobbies are given. In one of the easiest cases, e.g. maximum 3 hobbies and minimum 2 hobbies the depth-first search is building a solution relative fast. But the time to obtain the solution is increasing exponentially with the number of guests and chairs. For example, for a sample of initial data, on a PIV at 2.2 GHz with 512 Mb RAM, the problem for 64 guests is solved in 7 seconds, the one for 128 guests in 110 seconds, while the one for 256 guests in 1801 seconds. If the problem is more complicated, e.g. the minimum number of hobbies is 1, the depth-first search explores several branches of the search tree until it reach a solution. For example, for another sample of initial data generated with maximum 2 hobbies and minimum 1 hobbies, the problem for 64 guests is solved in 2103 seconds, while the one for 128 guests in 7361 seconds.

The solution process is divided into p tasks as follows. It is assumed that $2p$ divides the number of guests.

In a preprocessing phase of the initial data, the data set is split into p equal fragments. Then a search is performed looking if there are at least p special guests having the same sex and the maximal number of hobbies. If the answer is yes, they are distributed each to a distinct data fragment (interchanges are possible). If no, take the ones with the most close number to the maximal number of hobbies.

Each task receives a data fragment and the lower and the upper numbers of the seats that will be treated. The special guest selected in the first phase is seating on the first chair assigned to the task which treats the data fragment. The special guest is communicated to the task which treats the left neighbor data fragment. Same rules are applied for the depth-first search on each task, but on the different data fragment. The task

TABLE 4.1
Run time improvement using Parallel Jess for Miss Manners (256 guests, max 3 and min 2 hobbies) running on one computer

No. of instances	Running time	No. of fired rules/ instance
1	2321 seconds	33406 rules
2	214 seconds	8510 rules
4	35 seconds	2206 rules
8	11 seconds	590 rules

search is complete only if at least one hobby of the guest seating on the last chair assigned to task is on the list of the special guest communicated by the task treating the right neighbor fragment of data. Finally the pairs guest-chair provided by each task are collected by the master instance.

The above mentioned Jess source was modified accordingly. A batch file was written to launch the p Jess instance, similar to the one from Figure 3.1. Each instance loads the modified Jess source and according to its instance identifier processes a fragment of the data and send and/or receive the information about the special guests sitting on the chairs nearby the ones treated by the Jess instance.

The cluster environment used in the experiments consists of 8 IBM PCs at 1.5 GHz and 256 Mb RAM connected by a Myrinet switch with a peak speed for communications of 2Gb/s. When several instances of Jess are running on one of the cluster PCs, the solution is provided faster than in the case of using only one instance—Table 4.1 proves this fact. Moreover Table 4.2 shows that due the fact that the communication is needed only at the beginning and at the ending of the above described tasks, the parallel implementation efficiency is expected to be near to the ideal value.

5. P-systems and the available simulators. In the area of biology-inspired computing, a recently introduced model has taken inspiration from the structure and the functioning of living cells: Păun in [31] proposed an abstraction of the cell architecture and the way biological substances are modified or moved among compartments. Each compartment, delimited and separated from the rest by a membrane, can be seen as a computing unit having its own data and its local program (reactions). All compartments considered as a whole (the cell) can be seen as an unconventional computing device characterized by a membrane structure, where membranes can be hierarchically placed inside a unique external membrane delimiting the entire cell. All membranes are semi-permeable barriers, which either allow some substances to move in or out and consequently change their location in the membrane structure, or block the movement of some other substances.

This cell interpretation has its mathematical formalization in P systems, also called membrane systems, where a membrane structure can be described by a finite string of well matching parentheses. The substances and reactions are represented by objects and evolution rules. Objects are described as symbols or strings over a given alphabet, evolution rules are given as rewriting rules. The rules act on objects, by modifying and moving them, and they can also affect the membrane structure, by dissolving or dividing the membranes. A computation in P systems is obtained by starting from an initial configuration, identified by the membrane structure, the objects and the rules initially present inside it, and then letting the system to evolve. The application of rules is performed in a nondeterministic and maximal parallel manner: all the applicable rules have to be used to modify all objects which can be the subject of a rule, and this is done in parallel for all membranes. A universal clock is assumed to exist. The computation halts when no rule can be further applied. The output is defined in terms of the objects sent out to the external membrane or collected inside a specified membrane.

An example of the mathematical representation of a P-system is given in Figure 5.1. The different variants of P systems found in the literature are generally thought as generating devices. Almost all implementations of P system simulators are considering deterministic P systems.

TABLE 4.2
Run time improvement (and speedup) using the cluster nodes

No. of instances	1	2		4		8	
	Time	Time	S_2	Time	S_4	Time	S_8
1	2321 s	-	-	-	-	-	-
2	214 s	112 s	1.91	-	-	-	-
4	35 s	19 s	1.84	11 s	3.18	-	-
8	11 s	6 s	1.83	4 s	2.75	3 s	3.67

In the following experiments we have considered the particular P system described in [32] and in Figure 5.2 to solve the validity problem—given a boolean formula in conjunctive normal form, to determine whether or not it is a tautology. If we consider the problem input in the form

$$\wedge_{i=1}^{m} \vee_{j=1}^{k_i} x_{ij} \quad \text{where } x_{ij} \in \{X_1, \ldots, X_n, \overline{X}_1, \ldots, \overline{X}_n\}$$

the P system solves the NP-complete problem in $5n + 2m + 4$ evolution steps (in the external membrane it is obtained 'Yes' or 'No'). The number of membranes (the P system degree) increases by division from only 3 initially (two internal ones, plus the external one) to $2^n + 2$ at the computation end. This example is of particular interest for parallel simulation using dynamic task creation.

In the attempt to implement membrane computing on the usual computer one needs to simulate nondeterminism on a deterministic machine.

There are several attempts to simulate P systems on the existing sequential computers: a Visual C++ simulation for P systems with active membranes and catalytic P systems allowing graphical simulation and step-by-step observations of the membrane system behavior is reported in [10]; a Java implementation is reported in [28]; an implementation of transition P systems in Haskell is discussed in [3, 5]; another implementation of transition P system was done in MzScheme [4]; rewriting P systems and P systems with symport/antiport rules were described as executable specifications in Maude in [2, 39]; in [36] the membrane system is programmed in VHDL; an implementation of Cayley P Systems was written in MGS [21]; transition P systems and deterministic P systems with active membranes were simulated also in Prolog [14, 15, 37]; recognizer P systems with active membranes, input membrane and external output were simulated in Clips and used to solve two NP-complete problems in [32, 33, 34].

By simulating parallelism and nondeterminism on a sequential machine one can lose the power and attractiveness of P system computing. Therefore the simulations on multiple processors are useful. A parallel and a cluster implementation for transition P systems in C++ and MPI were reported in [9, 11, 12]. A distributed implementation based on Java RMI was also described in [42].

Unfortunately, the above mentioned simulators are capable to handle only P systems solving small problems. The computational power of a P system is not proved by small problems for which we already have faster algorithms, but by large problems where the NP is becoming an issue. Surely the space expansion requested by a P system evolution is a big problem, but a step forward is to use a large amount of computing devices in the simulation.

6. Using Parallel Jess in a P system simulation. We consider that the existing Clips implementations (compatible with Jess) are written in a language close to the mathematical description of the P systems. Also they provide a natural internal mechanism allowing multiple rule firing, and they are easily adaptable to different variants of P systems.

In this context, two developing directions were foreseen: the first one deals with the simulator improvement by using multiple computing units (reported here), and the second one deals with the improvement of the Clips implementation allowing wider set of P system variants to be simulated, queries strategies for P system status, objects, membranes, or rules, and an easy to use interface for describing and studying new P systems (reported in [6]). Hopefully the two improvements will lead to a faster P system simulator than the current available ones.

P-system: $(\Sigma, H, \mu, \omega_1, \ldots, \omega_N, R)$
N: the number of initial membranes (system degree)
Σ: the alphabet of symbol-objects
H: finite set of membrane labels
μ: the membrane structure
$\omega_1, \ldots, \omega_N$: strings over Σ, the initial multisets of objects, placed in each membrane of μ
R: a finite set of evolution rules of the following forms:
–object evolution rules (inside evolution): $[a \to b]_h^p$
–send-in rules (incoming objects): $a[]_h^{p_1} \to [b]_h^{p_2}$
–send-out rules (outgoing objects): $[a]_h^{p_1} \to b[]_h^{p_2}$
–dissolution rules (membrane disolved): $[a]_h^p \to b$
–division rules (membrane multiplication): $[a]_h^{p_1} \to [b]_h^{p_2}[c]_h^{p_3}$
where $a, b, c \in \Sigma$, $h \in H$, $p, p_1, p_2, p_3 \in \{+, -, 0\}$

FIG. 5.1. *A generic P-system with active membranes*

$N = 2$
$\Sigma = \{x_{i,j}, \overline{x}_{ij} : 1 \le i \le m,\ 1 \le j \le n\} \cup \{c_k : 1 \le k \le m+1\} \cup \{d_k : 1 \le k \le 2n+2m+2\}$
$\cup \{e_k : 0 \le k \le 3\} \cup \{r_{ik} : 0 \le i \le m,\ 1 \le k \le 2n\} \cup \{\text{Yes, No}\}$
$H = \{1, 2\}$
$\mu = [[]_2]_1$
$\omega_1 = \{e_1\},\ \omega_2 = \{d_1, <\text{the input symbols}>\}$
$R = \{[d_k]_2^0 \to [d_k]_2^+ [d_k]_2^- : 1 \le k \le n\} \cup \{[x_{i1} \to r_{i1}]_2^+,\ [\overline{x}_{i1} \to r_{i1}]_2^-,\ [x_{i1} \to]_2^-,\ [\overline{x}_{i1} \to]_2^+ : 1 \le i \le m\} \cup \{[x_{ij} \to x_{i\,j-1}]_2^+,\ [x_{ij} \to x_{i\,j-1}]_2^-,\ [\overline{x}_{ij} \to \overline{x}_{i\,j-1}]_2^+, [\overline{x}_{ij} \to \overline{x}_{i\,j-1}]_2^- : 1 \le i \le m,\ 2 \le j \le n\} \cup \{[d_k]_2^+ \to d_k[]_2^0,\ [d_k]_2^- \to d_k[]_2^0, : 1 \le k \le n\} \cup \{d_k[]_2^0 \to [d_{k+1}]_2^0, : 1 \le k \le n-1\} \cup \{[r_{ik} \to r_{i\,k+1}]_2^0 : 1 \le i \le m,\ 1 \le k \le 2n-1\} \cup \{[d_k \to d_{k+1}]_1^0 : n \le k \le 3n-3\} \cup \{[d_{3n-2} \to d_{3n-1} e_0]_1^0,\ e_0[]_2^0 \to [c_1]_2^-,\ [d_{3n-1} \to d_{3n}]_2^0\} \cup \{[d_k \to d_{k+1}]_1^0 : 3n \le k \le 3n+2m\} \cup \{[r_{1\,2n}]_2^- \to r_{1\,2n}[]_2^+\} \cup \{[r_{i\,2n} \to r_{i-1\,2n}]_2^+ : 1 \le i \le m\} \cup \{r_{1\,2n}[]_2^+ \to [r_{0\,2n}]_2^-\} \cup \{[c_k \to c_{k+1}]_2^+ : 1 \le k \le m\} \cup \{[c_{m+1}]_2^- \to c_{m+1}[]_2^+,\ [c_{m+1}]_1^0 \to c_{m+1}[]_1^+,\ \{d_{3n+2m+1}[]_2^+ \to [d_{3n+2m+2}]_2^+,\ d_{3n+2m+1}[]_2^- \to [d_{3n+2m+2}]_2^-,\ [d_{3n+2m+2}]_2^- \to d_{3n+2m+2}[]_2^-,\ [d_{3n+2m+2}]_1^+ \to d_{3n+2m+2}[]_1^-,\ [d_{3n+2m+2}]_1^0 \to d_{3n+2m+2}[]_1^-\} \cup \{[e_k \to e_{k+1}]_1^+ : 1 \le k \le 2\} \cup \{[e_2]_1^+ \to Yes[]_1^+,\ [e_3]_1^- \to No[]_1^-,\ [e_1]_1^- \to No[]_1^-\}$

FIG. 5.2. *A P-system for solving the validity problem*

TABLE 6.1
Communication actions

Rule	Case	Membrane	Actions
Object evolution	$[a \to b]_h^p$	m	modify m
			send children(m) "ev" (a, b)
		child(m)	recv m "ev"
			modify m
Send-in	$a[]_h^{p_1} \to [b]_h^{p_2}$	m	modify m, father(m)
			send father(m) "in" (a)
		father(m)	recv m "in"
			modify father(m)
Send-out	$[a]_h^{p_1} \to b[]_h^{p_2}$	m	modify m, father(m)
			send father(m) "out" (b)
		father(m)	recv m "out"
			modify father(m)
Dissolve	$[a]_h^p \to b$	m	modify m, father(m), children(m)
			send all "ds" (m, father(m), children(m))
		all	recv "ds" all
			modify m, father(m), children(m)
Division	$[a]_h^{p_1} \to [b]_h^{p_2}[c]_h^{p_3}$	m	modify m, father(m), children(m)
			send all "dv" (m, father(m), children(m))
		all	recv "dv" all
			modify m, father(m), children(m)

The aim of the experiment presented here is to measure the efficiency of Parallel Jess in a cluster environment when it is applied to a particular problem, the one involving P systems for which task parallelism is easy to be detected.

Different membranes can be distributed on different machines of a cluster. They can evolve accordingly the object evolution rules independently. New rules are added to express the message exchange. The send-in and send-out rules are requesting message exchanges, one message containing the object which goes out or comes in the membrane. The membrane towards which an object is migrating must probe constantly the existence of a new incoming message from the children of that membrane. The father membranes must communicate towards their children membranes any change in their content which can affect the send-in or send-out rules to be fired in the children membranes. Table 6.1 specifies the communication actions to be taken.

The division or the dissolution of a membrane means a dynamical change of the communication structure. This problem can be solved by Parallel Jess but it is not implemented yet in the Jess-based P system simulator.

We resumed the experiments reported in [32] describing partially a Clips simulator and a solution to the validity problem using P-systems. Starting from the Clips code, a Jess code for the simulator was written.

The numbers of rules to be fired are similar to those from the classical production system benchmarks like [30]. Table 6.2 specifies the number of rules to be fired in the case of checking the validity of a boolean expression with $m = n$. The requested time is measured on a machine of the cluster mentioned in the previous section. We concentrate our attention on the part of the simulation after the final division occurs in the P

TABLE 6.2
Test problem dimension

$m \times n$	Membranes	Rules fired after the last division	Time for firing those rules	Reaching the full membrane configuration	Total time of simulation
2×2	6	335	1 s	1 s	2 s
3×3	10	779	13 s	3 s	16 s
4×4	18	1835	428	6 s	434 s
5×5	34	?	Out of memory	49 s	?

```
MASTER

(batch ‘‘ParJess.clp")
(defglobal ?*t*=(time))
(connection)
(kernels 4)
(send -1 1 ‘‘(batch \"W\")"
(recv 1 2)
(stop)
(-(time) ?*t*)
```

```
WORKER (W)

 (batch ‘‘simulator")
(load-facts ‘‘Psyst")
(initialize)
(run)
```

FIG. 6.1. *Master and workers batch files in the case of a P-system and 4 workers*

system (the most consuming part of the simulation), i. e. when we have already $2^n + 2$ membranes.

The Parallel Jess code activated by the user is very simple and it is depicted in Figure 6.1. Table 6.3 describes shortly the code added in order to ensure the correct distribution and evolution of the membranes.

TABLE 6.3
Changes in the simulator code

What	How
Membrane template	New slot ‘owner’
Evolution, send-in/out rules	Fired only if the Jess instance owns the membrane
Send-in rule	Send to the father the object to be erased from its content
Send-out rule	Send to the father the object to be added to its content
New phase: send-recv	After evolution, send-in-out, messages are received
New rule: send-father-status	If the content of the father was changed, the new content is send to all membrane children
New rule: recv-father-status	As response to an incoming message indicating the father change, the local copy of the father is changed
New rule: recv-child-erase	As response to an incoming message indicating the child change, the incoming object is erased from father content
New rule: recv-child-add	As response to an incoming message indicating the child change, the incoming object is added to the father content

Each Jess instance reads the simulator rules, all the facts (the particular rules to be applied) and the membrane structure and contents. Each membrane is owned by a Jess instance. An instance can own one or several membranes. Rules are fired by the instance only if they are referring to an owned membrane. Each instance has copies of other instance membranes, with the actual content or an old one. A change in the content of the father membrane can lead to a change in the agendas of the membrane children. Sources and destination numbers used in the send and receive commands are referring to the membrane owners. The receiving rules do not have a blocking effect, unless the membrane receiver cannot evolve further without any incoming messages: the message arrival is tested again for different kinds of messages (due to evolution, send-in or send-out rules) until a message is received from another membrane owner or from the master instance. The send and receive are activated only if the two membranes involved in the exchange have different owners.

A significant time reduction of the running time of the P system simulator was obtained using the Parallel Jess version running on only one machine. Table 6.4 shows some examples: the shorter time is underlined. The number of Jess instances working concurrently on the same machine and leading to the lowest simulation time depends on the problem dimension. It seems that, at least in the test cases, for a $m \times m$ problem it is recommended to use m working Jess instances. To explain this phenomena, we remember that the basic

TABLE 6.4
Simulation time: the Jess instances are running on one machine

Membranes	Instances			
	1	2	3	4
6	1 s	1 s	3 s	3 s
10	13 s	5 s	5 s	10 s
18	428 s	45 s	25 s	33 s
34	Memory out	1054 s	325 s	200 s

rules have been changed: they can be fired only if the Jess instance owns the membrane. So the time for the matching process is considerably lower in each instance than in the case when only one Jess instance treats all the membranes.

A small time variation was registered when the membrane distribution to different instances was changed.

TABLE 6.5
Simulation time, speedup and efficiency: the Jess instances are running on different machines of the cluster

Instances	Machines	Membranes			
		6	10	18	34
1	1	1 s	13 s	428 s	Mem. out
2	1	1 s	5 s	45 s	1054 s
	2	1 s	3 s	23 s	528 s
	S_2	1	1.7	1.9	2
	E_2	0.51%	0.65%	0.95%	0.99%
3	1	3 s	5 s	25 s	325 s
	3	3 s	2 s	10 s	126 s
	S_3	1	2.5	2.5	2.6
	E_3	0.33%	0.83%	0.83%	0.87%
4	1	3 s	10 s	33 s	200 s
	4	2 s	3 s	9 s	52 s
	S_4	1.5	3.3	3.7	3.8
	E_4	0.37%	0.82%	0.91%	0.96%

Further reduction of the simulation time is expected when the Jess instances are running on different machines. This expectation is confirmed by the tests. Table 6.5 refers to some of them. The speedup S_p and the efficiency E_p of the parallel implementation are registered in this table. Those values are close to the ideal ones. It is easy to see the normal increase of the speedup with the number of rules to be fired. We expect to obtain better results for larger dimension for the validity problem.

7. Conclusions and further improvements. Several approaches to construct parallel rule-based engines were discussed in this paper. Following one approach, we have initiated the development of Parallel Jess, a wrapper for Jess enabling it to cooperate with other Jess instances running in a cluster environment.

At this stage, Parallel Jess exists as a demo system. Changing the message passing interface from JPVM to MPI will allow the system migration towards Grids to connect more than one cluster running several Jess instances.

A version for parallel Clips will be soon derived using the same wrapping model. A first step to do that has been already undertaken: to enrich Clips with a socket communication facility.

Further extensions to other languages for production systems is the subject for discussions.

We demonstrated the efficiency of Parallel Jess on a concrete application involving P systems. The P-system simulator using the Parallel Jess will be further developed to include facilities for membrane dissolution and division; tests are necessary to compare the current parallel implementation with other existing ones.

Further experiments are needed, not necessarily related to P systems.

REFERENCES

[1] J. N. AMARAL, *A parallel architecture for serializable production systems*, Ph.D. Thesis, University of Texas, Austin, 1994, available at `ftp://ftp.capsl.udel.edu/pub/people/amaral/tese.ps.gz`

[2] O. ANDREI, G. CIOBANU AND D. LUCANU, *Rewriting P systems in Maude*, Pre-procs. 5th Workshop on Membrane Computing, WMC5, Milano, Italy, 2004, available at `http://psystems.disco.unimib.it/procwmc5.html`

[3] F. Arroyo, C. Luengo, A. V. Baranda and L. F. de Mingo, *A software simulation of transition P systems in Haskell*, in Procs. 3rd Workshop on Membrane Computing WMC2, Curtea de Arges, Romania, 2002, Gh. Paun et al. (eds.), Springer, LNCS 2597 (2003), pp. 19–32.

[4] D. Balbontin Noval, M. J. Perez-Jimenez, F. Sancho-Caparrini, *A MzScheme implementation of transition P systems*, in Procs. 3th Workshop on Membrane Computing, WMC2, Curtea de Arges, Romania, 2002, Gh. Paun et al. (eds.), Springer, LNCS 2597 (2003), pp. 58–73.

[5] A. V. Baranda, F. Arroyo, J. Castellanos, R. Gonzalo, *Towards an electronic implementation of membrane computing: A formal description of non-deterministic evolution in transition P systems*, in Procs. 7th Workshop on DNA Based Computers, Tampa, Florida, 2001, N. Jonoska and N.C. Seeman (eds.), Springer, LNCS 2340 (2002), pp. 350–359.

[6] C. Bonchiş, G. Ciobanu, C. Izbaşa and D. Petcu, *A Web-based P systems simulator and its parallelization*, in Procs. UC 2005, C.S. Calude et al. (Eds.), LNCS 3699 (2005), pp. 58–69.

[7] C. S. Calude and G. Păun, *Computing with cells and atoms: after five years*, CDMTCS Research Report Series, CDMTCS-246, 2004, available at `http://www.cs.auckland.ac.nz/CDMTCS/researchreports/246cris.pdf`

[8] Y. Cengeloglu, S. Khajenoori and D. Linton, *A framework for dynamic knowledge exchange among intelligent agents*, in Procs. AAAI Symposium, 1994.

[9] G. Ciobanu, R. Desai and A. Kumar, *Membrane systems and distributed computing*, in Membrane Computing, Procs. WMC2, Curtea de Arges, Romania, 2002, Gh. Paun et al. (eds.), Springer, LNCS 2597 (2003), pp. 187–202.

[10] G. Ciobanu and D. Paraschiv, *Membrane software. A P system simulator*, Fundamenta Informaticae 49, no. 1-3 (2002), pp. 61–66.

[11] G. Ciobanu and G. Wenyuan, *A parallel implementation of the transition P systems*, in Pre-procs. 4th Workshop on Membrane Computing, Taragona, Spain (2003), pp. 169–184.

[12] G. Ciobanu and G. Wenyuan, *P systems running on a cluster of computers*, in Procs. 4th Workshop on Membrane Computing, WMC3, Taragona, Spain, 2003, C. Martin-Vide et al. (eds.), Springer, LNCS 2933 (2004), pp. 123-139.

[13] *Clips, A tool for building expert systems*, available at `http://www.ghg.net/clips/Clips.html`

[14] A. Cordon-Franco, M. A. Gutierrez-Naranjo, M. J. Perez-Jimenez and F. SanchoCaparrini, *A Prolog simulator for deterministic P systems with active membranes*, in Procs. 1st Brainstorming Week on Membrane Computing, Taragona, Spain (2003), pp. 141–154.

[15] A. Cordon-Franco, M. A. Gutierrez-Naranjo, M. J. Perez-Jimenez, A. Riscos-Nunez and F. Sancho-Caparrini, *Implementing in Prolog an effective cellular solution for the Knapsack problem*, in Procs. 4th Workshop on Membrane Computing, WMC3, Taragona, Spain, 2003, C. Martin-Vide et al. (eds.), Springer, LNCS 2933 (2004), pp. 140–152.

[16] D. Fezzani, J. Desbiens, *Epidaure: A Java distributed tool for building DAI applications*, in Procs. EuroPar'99, P.Amestoy et al (eds), Springer, LNCS 1685 (1999), pp. 785–789.

[17] E. Friedman-Hill, *Jess in action: rule-based systems in Java*, Manning Publications, 2003.

[18] E. Friedman-Hill, *Jess: manners and waltz test for Jess* (2003), available at `http://www.mail-archive.com/jess-users@sandia.gov/msg05113.html`

[19] C. L. Forgy, *Rete: A fast algorithm for the many pattern/many object pattern match problem*, Artificial Intelligence 19 (1982), pp. 17–37.

[20] D. Gagne, A. Garant, *DAI-Clips: Distributed, asynchronous, interacting Clips*, Procs. Clips'94 (electronic version), 3rd Conf.on Clips, Lyndon B. Johnson Space Center (1994), pp. 297–306, available at `http://www.ghg.net/clips/Clips.html`

[21] J. L. Giavitto, O. Michel, J. Cohen, *Accretive rules in Cayley P systems*, in Procs. 3th Worshop on Membrane Computing, WMC2, Curtea de Arges, Romania, 2002, Gh. Paun et al. (eds.), Springer, LNCS, 2597 (2003), pp. 319-338.

[22] L. O. Hall, B. H. Bennett, I. Tello, *PClips: Parallel Clips*, Procs. Clips'94 (electronic version), 3rd Conf.on Clips, Lyndon B. Johnson Space Center, 1994, pp. 307–314, available at `http://www.ghg.net/clips/Clips.html`

[23] *Jess, the rule engine for Java platform*, available at `http://herzberg.ca.sandia.gov/jess/`

[24] *JPVM, The Java Parallel Virtual Machine*, 1999, available at `http://www.cs.virginia.edu/~ajf2j/jpvm.html`

[25] P. Y. Li, *dClips: A distributed Clips implementation*, Procs. AIAA Computing in Aerospace Conference, San Diego, 1993, AIAA Paper 93-4502-CP.

[26] R. Miller, *PClips: A distributed expert system environment*, Procs. Clips Users Group Conf, 1990.

[27] L. Myers, K. Pohl, *Using PVM to host Clips in distributed environments*, Procs. Clips'94 (electronic version), 3rd Conf.on Clips, Lyndon B. Johnson Space Center, 1994, pp. 177–186, available at `http://www.ghg.net/clips/Clips.html`

[28] I. A. Nepomuceno-Chamorro, *A Java simulator for basic transition P systems*, in Procs. 2nd Brainstorming Week on Membrane Computing, Sevilla, Spain, 2004, available at `http://www.gcn.us.es/Brain/bravolpdf/SIM21.pdf`

[29] L. O'Higgins Hall, L. Prasad, E. Jackson, *Parallel Clips for current hypercube architectures*, Procs. FLAIRS'93, Ft. Lauderdale, 1993.

[30] *OPS5 benchmark suite*, available at `http://www.cs.utexas.edu/ftp/pub/ops5-benchmark-suite/` (1993), and `http://www.pst.com/benchcr2.htm` (2003).

[31] G. Păun, *Computing with membranes*, in Journal of Computer and System Sciences, 61, pp. 108–143, 2000 and TUCS Report No. 208, 1998.

[32] M.J. Perez-Jimenez, F.J. Romero-Campero, *A Clips simulator for recognizer P systems with active membranes*, 2002, available at `http://www.gcn.us.es/Brain/bravolpdf/Clips.pdf`

[33] M. J. Perez-Jimenez, F. J. Romero-Campero, *Solving the BinPacking problem by recognizer P systems with active membranes*, in Procs. 2nd Brainstorming Week on Membrane Computing, 2004, Sevilla, Spain, Gh. Paun et al (eds.) (2004), pp. 414–430.

[34] M. J. Perez-Jimenez, F. J. Romero-Campero, *Trading polarizations for bi-stable catalysts in P systems with active membranes*, in Pre-procs. 5th Workshop on Membrane Computing WMC5, Milano, Italy, (2004), pp. 327–342.

[35] D. Petcu, *Parallel Jess*, in Procs. ISPDC 2005, 4-6 July 2005, Lille, IEEE Computer Society Press, Los Alamitos (2005), pp. 307–314.

[36] B. Petreska, C. Teuscher, *A reconfigurable hardware membrane system*, in Procs. 4th Workshop on Membrane Computing, WMC3, Taragona, Spain, 2003 C. Martin-Vide et al (eds.), Springer, LNCS 2933 (2004), pp. 269–285.
[37] P. Prakash Mohan, *Computing with membranes*, 2001, available at `http://psystems.disco.unimib.it/download/rep.zip`
[38] *P systems Web page*, 2004, available at `http://psystems.disco.unimib.it`
[39] Z. Qi, C. Fu, D. Shi and J. You, *Specification and execution of P systems with symport/ antiport rules using rewriting logic*, Pre-procs. 5th Workshop on Membrane Computing, WMC5, Milano, Italy, (2004).
[40] G. Riley, *Implementing Clips on a parallel computer*, Procs. SOAR '87, NASA/Johnson Space Center, Houston, TX, 1987.
[41] R. D. Schimkat, W. Blochinger, C. Sinz, M. Friedrich and W. Küchlin, *A service-based agent framework for distributed symbolic computation*, in Procs. HPCN 2000, M. Bubak et al (eds.), LNCS 1823 (2000), pp. 644–656.
[42] A. Syropoulos, E. G. Mamatas, P. C. Allilomes and K. T. Sotiriades, *A distributed simulation of transition P systems*, in Procs. 4th Workshop on Membrane Computing, Taragona, Spain, 2003, C. Martin-Vide et al. (eds.), Springer, LNCS 2933 (2004), pp. 357–368.
[43] S. Wu, D. Miranker and J. Browne, *Towards semantic-based exploration of parallelism in production systems*, TR-94-23, 1994, available at `http://historical.ncstrl.org/litesite-data/utexas_cs/tr94-23.ps.Z`

Edited by: M. Tudruj, R. Olejnik.
Received: February 24, 2006.
Accepted: July 30, 2006.