On the Simulations of Evolution-Communication P Systems with Energy without Antiport Rules for GPUs

Richelle Ann B. Juayong[1], Francis George C. Cabarle[1], Henry N. Adorna[1], Miguel A. Martínez–del–Amor[2]

[1] Algorithms & Complexity Lab
Department of Computer Science
University of the Philippines Diliman
Diliman 1101 Quezon City, Philippines
E-mail: rbjuayong@up.edu.ph, fccabarle@up.edu.ph, hnadorna@dcs.upd.edu.ph

[2] Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Seville
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
E-mail: mdelamor@us.es

Summary. In this report, we present our initial proposal on simulating computations on a restricted variant of Evolution-Communication P system with energy (ECPe system) which will then be implemented in Graphics Processing Units (GPUs). This ECPe systems variant prohibits the use of antiport rules for communication. Several possible levels of parallelizations for simulating ECPe systems computations on GPUs are emphasized. Our work is based on a localized matrix representation for the mentioned variant given in a previous literature. Our proposal employs a methodology for forward computing also discussed in the said literature.

Key words: Membrane computing, Parallel computing, GPU computing

1 Introduction

Evolution-Communication P systems with energy (ECPe systems) [1] is a variant of P systems introduced in 2009 to initiate a framework for communication complexity. It originates from Evolution-Communication P (ECP) systems [10], a hybrid of two well-investigated variants, Transition P systems [9] and P systems with Symport and Antiport rules [11]. The difference between ECPe and ECP systems is the presence of a special object called 'energy' in the former, which can be produced through evolution rules and consumed in communication rules. One

crucial restriction for ECPe systems includes the use of energy for each communication rule. Thus, no object can be communicated without using some 'quanta' of energy. Moreover, upon being delivered to a receiving region, the energy used in a communication does not pass through any membrane. In this manner, it is said that the energy used in the process of communication is being 'lost'.

Several recent works have introduced the concept of representing certain P system variants and their computations as matrices and vector-matrix operations, respectively. In particular, variants known as Spiking Neural P (SNP) systems and their matrix representations were introduced in [5] whereas matrix representations for ECPe systems were given in [6]. Aside from creating a 'convenient' and relatively compact way of describing the systems and their computations, the matrix representations add additional ease to their simulation and implementation in parallel hardware. Vector and matrix operations are highly parallelizable and can be efficiently implemented in parallel hardware, including Graphics Processing Units (GPUs). GPUs are massively parallel hardware not like current generation CPUs. Using their respective matrix representations, SNP systems have been successfully implemented in GPUs in [2] and more recently in [3]. The intention of this current work is to continue such trend i.e. to provide our methodology on how to implement ECPe systems computations (using the vector-matrix representations) on parallel hardware, in particular GPUs.

2 Evolution-Communication P Systems with Energy

2.1 Formal Definition of ECPe systems

Before we proceed, we note that the readers are assumed to be familiar with the fundamentals of formal language theory and membrane computing [9].

A relatively new variant of Evolution-Communication P systems [10] has been introduced in [1] to evaluate communication that is dependent on some energy produced from evolution rules. A special object e is introduced to the system to represent a quantum of energy. We use the definition for EC P system with energy (ECPe system) from [1].

Definition 1. *An EC P system with energy is a construct of the form*

$$\Pi = (O, e, \mu, w_1, \ldots, w_m, R_1, R'_1, \ldots, R_m, R'_m, i_{out})$$

where:

(i) m pertains to the total number of membranes;
(ii) O is the alphabet of objects;
(iii) μ is the membrane structure which can be denoted by a set of paired square brackets with labels. We say that membrane i is the *parent membrane* of a membrane j, denoted $parent(j)$, if the paired square brackets representing

membrane j is located inside the paired square brackets representing membrane i, i.e. $[_i \ldots [_j \,]_j]_i$. Reversely, we say that membrane j is a *child membrane* of membrane i, denoted $j \in children(i)$ where $children(i)$ refers to the set of membranes contained in membrane i. The relation of parent and child membrane becomes more apparent when we represent the membrane structure as a tree. Since order does not matter in our model, there can be multiple trees (isomorphic with respect to children of a node), each corresponding to the same membrane structure representation.

(iv) $w_1, \ldots, w_m$ are strings over O^* where w_i denotes the multiset of object present in the region bounded by membrane i.

(v) $R_1, \ldots, R_m$ are sets of evolution rules, each associated with a region delimited by a membrane in μ;

- An evolution rule is of the form $a \to v$ where $a \in O$, $v \in (O \cup \{e\})^*$. In the event that this type of rule is applied, the object a transforms into a multiset of objects v in the next time step. Through evolution rules, object e can be produced, but e should never be in the initial configuration and object e is not allowed to evolve.

(vi) $R'_1, \ldots, R'_m$ are sets of communication rules, each associated with a membrane in μ; A communication rule can either be a symport or an antiport rule:

- A symport rule can be of the form (ae^i, in) or (ae^i, out), where $a \in O$, $i \geq 1$. By using this rule, i copy of e objects are consumed to transport object a inside (denoted by in) or outside (denoted by out) the membrane where the rule is defined. To consume copies of object e means that upon completion of the transportation of object a, the occurrences of e are lost, they do not pass from a region to another one.
- An antiport rule is of the form $(ae^i, out; be^j, in)$ where $a, b \in O$ and $i, j \geq 1$. By using this rule, we know that there exists an object a in the region immediately outside the membrane where the rule is declared, and an object b inside the region bounded by the membrane. In the application of this rule, object a and object b are swapped using i and j copies of object e in the different regions, respectively. As in symport rules, the copies of object e are lost after the application.

We say that a communication rule has a *sending* and *receiving* region. For a rule $r \in R'_i$ associated with an in label, its receiving region is region i and its sending region is the $parent(i)$. On the other hand, the sending and receiving regions are reversed for a rule $r \in R'_i$ associated with an out label. For an antiport rule $r \in R'_i$, region i and $parent(i)$ are both sending and receiving region. Also, note that no communication can be applied without the utilization of object e.

(vii) $i_{out} \in \{0, 1, \ldots, m\}$ is the output membrane. If $i_{out} = 0$, this means that the environment shall be the placeholder of the output.

Rules are applied in a nondeterministic, maximally parallel manner. Nondeterminism, in this case, has the following meaning: when there are more than two evolution rules that can be applied to an object, the system will randomly choose

the rule to be applied for each copy of the object. The system assumes a universal clock for simultaneous processing of membranes; all applicable rules have to be applied to all possible objects at the same time. The behavior of maximally parallel application of rule requires that all object that can evolve (or be transferred) should evolve (or be transferred).

Note that there is a one-to-one mapping between region and membrane, however, strictly, region refers to the area delimited by a membrane. A configuration at any time i, denoted by C_i, is the state of the system; it consists of the membrane structure and the multiset of objects within each membrane. A transition from C_i to C_{i+1} through nondeterministic and maximally parallel manner of rule application can be denoted as $C_i \Rightarrow C_{i+1}$. A series of transition is said to be a computation and can be denoted as $C_i \Rightarrow^* C_j$ where $i < j$. Computation succeeds when the system halts; this occurs when the system reaches a configuration wherein none of the rules can be applied. This configuration is called a halting configuration. If there is no halting configuration—that is, if the system does not halt—computation fails, because the system did not produce any output. Output can either be in the form of objects sent outside the skin, the outermost membrane, or objects sent into an output membrane.

We let $N(\Pi)$ be the set of numbers generated by a given ECPe system Π.

2.2 An Example

To show how ECPe system works, we shall give an example of an ECPe system with two membranes adapted from [1]:

$$\Pi = (\{a, \#\}, e, [_1[_2]_2]_1, a^2\#, \lambda, \{r_{11} : a \to aa, r_{12} : a \to ee\}, \emptyset,$$
$$\{r_{21} : \# \to \#\}, \{r'_{21} : (ae, in), r'_{22} : (\#e, in)\}, 2)$$

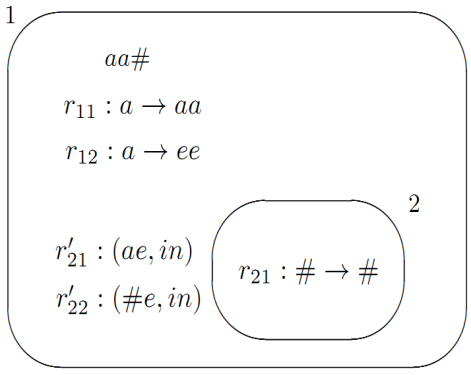

Fig. 1. Graphical representation of an ECPe system Π where $N(\Pi) = \{2(n+1)|n \geq 0\}$ from [1].

A graphical illustration of Π is shown in Figure 1. Its output is $N(\Pi) = \{2(n+1)|n \geq 0\}$. The computation to generate this proceeds as follows:

Initially, we can use either rule r_{11} or r_{12} in order to consume the copies of object a in membrane 1. At any time, we can use both r_{11} and r_{12} to evolve copies of object a. For every copy of a, we produce either two copies of object e or another two copies of object a, therefore, we are always assured that multiplicity of object a in region 1 is even, as well as the multiplicity of object e. Note that upon introduction of object e in the system, it should immediately be used (in the next step) to transport copies of object a in region 2, otherwise, it will be used to transport the trap symbol # in region 2 using rule r'_{22} leading the system to a never ending computation due to rule r_{21}. For a computation to halt, rule r'_{21} should be the last rule to be applied. Since copies of object e transporting copies of object a in region 2 is always even, we are assured that the multiplicity of objects (copies of object a) in region 2 is also even. The minimum value 2 is produced when we use both rule r_{11} and r_{12} in configuration C_1. In the next step, we use two applications of rule r'_{21}. This will cause the system to halt.

2.3 Representation and Methodology for Forward Computing

In this section, we relate how computations in ECPe systems without antiport rules can be performed in a localized manner. As will be shown, when we do not allow antiport rules, membranes can compute more independently. This representation has been used in [4] to answer the problem of computing backward and forward. We shall relate the methodology for the latter. By computing forward, the problem is to find the next configurations that can be yielded in one computational step given a current configuration.

Let $h \in \{0, 1, 2, \ldots, m\}$ where region 0 refers to the region located outside the skin, the outermost membrane. The following notations and definitions are adopted from [4] and used in the remaining parts of this section.

- Let $IO(r, h)$ be the set of objects in region h involved in a rule r.
- Let $TO(r, h)$ be the set of objects in region h that trigger a rule r.
- The set of rules $IR(h) = R_h \cup R'_h \cup (\bigcup_{h' \in children(h)} R'_{h'})$ represents the set of rules that directly influences the content of region h at any time of a computation. An object α in region h at any time $i \geq 0$ may either be produced by an evolution rule, transported from neighboring region to region h (or vice versa), or simply carried over. In the first scenario, it is by definition that the rule that produced object α must be in R_h. A neighboring region may either be a region delimited by $parent(h)$ or regions delimited by membranes in $children(h)$. In the first case, the rules for communication are in R'_h while in the second case, the rules for communication must be in one of $R'_{h'}$ where $h' \in children(h)$.
- The set $PO(h) = \{\alpha|\alpha \text{ appeared in } w_h\} \cup (\bigcup_{r \in IR(h)} IO(r, h))$ represents the set of objects (including special object e) that may possibly occur in region h at any time of a computation. Originally, the objects that surely exist in the region are the elements present in w_h. In order to create a copy of an object

α, object α must either be produced or transported in region h through rules in $IR(h)$.

- The set $TR(h) = \{r | TO(r,h) \neq \emptyset\}$ corresponds to the set of rules that contribute to the decrease of objects in region h. In order to activate rules belonging to such set, there must be a trigger object that may either be consumed or be used for transportation

In order to represent configuration and rule application in terms of vectors, and represent effect of a rule in each region using a matrix, the concept of total order must be utilized. We note that for all the vector (and matrix) representation constructed in the remaining parts of this section, there is a need to define a *total order* $\langle p_1, p_2, ... \rangle$ (so that p_i is considered the i^{th} element in a defined set) over the elements involved in the column for vectors (rows and columns for matrices). As can be observed, this is used so that elements are uniquely identified by their positions in the order to where they belong to and to assure that the position of elements are correct during the vector-matrix operation.

Definition 2. ***Configuration Vector for each Region*** h
A configuration vector $\mathbf{C}_{i,h}$ *is a vector whose length is* $|PO(h)|$. *The vector* $\mathbf{C}_{i,h}(\alpha)$ *refers to the multiplicity of object* α *in region* h *at configuration* C_i.

Definition 3. ***Application Vector for each Region*** h
An application vector $\mathbf{a}_{i,h}$ *is a vector whose length is* $|R(h)|$. *The vector* $\mathbf{a}_{i,h}(r)$ *refers to the number of application of rule* r *specifically in region* h *during the transition* $C_{i-1} \Rightarrow C_i$.

Definition 4. ***Transition Matrix for each Region*** h
A transition matrix $M_{\Pi_{ECPe},h}$ *is a matrix whose dimension is* $|R(h)| \times |PO(h)|$. *The matrix* $M_{\Pi_{ECPe},h}(r,\alpha)$ *returns the number of consumed or produced object* α *in region* h *upon single application of rule* r. *The consumed objects have negative values while the produced objects are positive. If object* α *in region* h *is not used in rule* r, *then its value is zero.*

Given application vector $a_{i,h}$ representing a maximal set of rule applications applied in a configuration C_{i-1} to achieve configuration C_i, the paper [4] showed that a transition $\mathbf{C}_{i-1} \rightarrow \mathbf{C_i}$ can be represented by performing

$$\mathbf{C}_{i,h} = \mathbf{C}_{i-1,h} + \mathbf{a}_{i,h}.M_{\Pi,h} \tag{1}$$

for each region h provided that if h and h' are the sender and receiver regions corresponding to a communication rule $r' \in IR(h) \cap IR(h')$, then $\mathbf{a}_{i,h}(r') = \mathbf{a}_{i,h'}(r')$.

Illustrating Localized Computation

To illustrate localized computation, we represent a possible transition $C_0 \Rightarrow C_1$ by showing the effect of applying rule r_{11} and rule r_{12} once on the initial configuration of the example presented in Section 2.2. Since $PO(0) = IR(0) = \emptyset$, the participation of the environment is not needed in any part of the computation.

For Region 1

At the onset, we can impose a total order $\langle a, \#, e\rangle$ over $PO(1)$ and total order $\langle r_{11}, r_{12}, r'_{21}, r'_{22}\rangle$ over $IR(1)$. The initial configuration will be represented by the configuration vector $\mathbf{C}_{0,1}$ where

$$\mathbf{C}_{0,1} = \begin{pmatrix} 2 & 1 & 0 \end{pmatrix}$$

and the representation for single application of both rules r_{11} and r_{12} will be given by application vector $\mathbf{a}_{1,1}$ where

$$\mathbf{a}_{1,1} = \begin{pmatrix} 1 & 1 & 0 & 0 \end{pmatrix}$$

Applying Equation (1) with the transition matrix $M_{\Pi_{ECPe},1}$ containing the values shown below:

$$M_{\Pi_{ECPe},1} = \begin{pmatrix} 1 & 0 & 0 \\ -1 & 0 & 2 \\ -1 & 0 & -1 \\ 0 & -1 & -1 \end{pmatrix}$$

will yield the configuration vector $\mathbf{C}_{1,1}$

$$\mathbf{C}_{1,1} = \begin{pmatrix} 2 & 1 & 2 \end{pmatrix}$$

which means that in the next configuration, there will be two copies of both object a and the special object e and a single copy of the trap symbol # in region 1.

For Region 2

For region 2, we impose total order $\langle a, \#\rangle$ over $PO(2)$ and total order $\langle r_{21}, r'_{21}, r'_{22}\rangle$ over $IR(2)$. Since initially, no objects are present in region 2 and the rules involved in the transition $C_0 \Rightarrow C_1$ are not in $IR(2)$, the configuration vector $\mathbf{C}_{0,2}$ and the application vector $\mathbf{a}_{1,2}$ will all contain zero values. We now show the transition matrix $M_{\Pi_{ECPe},2}$

$$M_{\Pi_{ECPe},2} = \begin{pmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}$$

Since application vector $\mathbf{a}_{1,2}$ is a zero vector, the configuration vector $\mathbf{C}_{1,2}$ remains also a zero vector.

Notice that all the declared communication rules influence the multiplicity of objects in both region 1 and region 2. However, region 1 contains negative values because it acts as a sending region while region 2 have non-negative values since it acts as a receiving region. Also, matrix $M_{\Pi_{ECPe},2}$ shows that the special object e can never reach region 2.

Forward Computing in ECPe systems without Antiport Rules

Shown below is a methodology for forward computing shown in [4].

1. Categorize all possible objects in $PO(h)$ for all region h.

First, all $\alpha \in PO(h)$ are categorized for a certain region h. These categories are:

- Category 1: Evolution Trigger
  Object α is an evolution trigger if there exists $r \in R_h$ such that $TO(r,h) = \{\alpha\}$.
- Category 2: Communication Trigger Only
  Object α belongs in this category if there does not exist $r \in R_h$ such that $TO(r,h) = \{\alpha\}$ but there exists $r' \in IR(h)$ such that $\alpha \in TO(r',h)$.
- Category 3: Not a Trigger
  Object α is neither in Category 1 nor in Category 2.

2. Construct identity rules for objects in Category 2 and 3 for all region h.

For each $\alpha \in PO(h)$ that falls under one of Category 2 and Category 3, an identity rule $\alpha \to \alpha$ is added. All these rules shall be contained in a set labelled $R_{add,h}$. Also, a list of $\alpha' \in PO(h) - \{e\}$ that fall under Category 2 is maintained, the list shall be labelled $List_{cat_2}$ and sorted $List_{cat_2}$ in increasing order of energy requirement for transport.

3. Construct Trigger Matrix $TM_{\Pi_{ECPe},h}$ for all region h

The defined rules represented in the rows of $TM_{\Pi_{ECPe},h}$ are the rules that contribute to the decrease of multiplicity of objects in region h. These rules are represented in the set $TR(h)$. The additional rules from $R_{add,h}$ are represented in the rows as well. The set of objects represented in the columns of $TM_{\Pi_{ECPe},h}$ is $PO(h)$. Therefore, $TM_{\Pi_{ECPe},h}$ has dimensions $|TR(h) \cup R_{add,h}| \times |PO(h)|$. $TM_{\Pi_{ECPe},h}(r,\alpha)$ returns the multiplicity of α in region h needed to activate a single application of rule r.

4. Set the dimension of the vector of unknowns (also called extended application vector) $\mathbf{a'}_{i,h}$ for all region h

The length of $\mathbf{a'}_{i,h}$ is $|TR(h) \cup R_{add,h}|$.

5. Solve system of linear equation

Find all solutions to the equation

$$\mathbf{a'}_{i,h}.TM_{\Pi_{ECPe},h} = \mathbf{C}_{i-1,h} \qquad (2)$$

Since elements of vector $\mathbf{a'}_{i,h}$ pertain to number of application of rules, these elements must be natural numbers. The value $\mathbf{a'}_{i,h}(r)$ can be interpreted as either the number of application of each rule $r \in TR(h)$ or how many object α is unevolved or unmoved (if $(r : \alpha \to \alpha) \in R_{add,h}$). Note that $TR(h)$ and $R_{add,h}$ are disjoint sets.

6. Filter solutions in Step 5

For each region h, if $List_{cat_2} \neq \emptyset$, scan the sorted $List_{cat_2}$ and find out the first object, labelled $\alpha_{cat_2,min}$, falling under Category 2 whose corresponding identity rule application is non-zero, i.e. $\mathbf{a'}_{i,h}(\alpha_{cat_2,min} \to \alpha_{cat_2,min}) > 0$. Since $List_{cat_2}$ is sorted increasingly according to transport energy requirement, the object $\alpha_{cat_2,min}$ has the minimum energy required for communication. Let its corresponding energy be labelled $energy(\alpha_{cat_2,min})$. Solutions are filtered in step 5 by adding, for each region h with a non-empty $List_{cat_2}$, the inequality below:

$$\mathbf{a'}_{i,h}(e \to e) < energy(\alpha_{cat_2,min}) \qquad (3)$$

7. Finding $\mathbf{a}_{i,h}$

Upon finding values for $\mathbf{a'}_{i,h}$ in all region h, all identity rules $r' \in R_{add,h}$ are omitted. The values of an application vector $\mathbf{a}_{i,h}$ are filled through the equation

$$\mathbf{a}_{i,h}(r) = \mathbf{a'}_{i,h}(r),\ r \in R_h \qquad (4)$$

For every communication rule $r \in IR(r, h') \cap IR(r, h'')$,

$$\mathbf{a}_{i,h'}(r) = \mathbf{a}_{i,h''}(r) = \mathbf{a'}_{i,h''}(r) \qquad (5)$$

where region h'' is the sending region of communication rule r.

An Illustration

We illustrate how we can compute forward in ECPe systems without antiport by using the ECPe system given in Section 2.2. We maintain the total orders $\langle a, \#, e\rangle$ over elements of $PO(1)$, $\langle a, \#\rangle$ over elements of $PO(2)$, $\langle r_{11}, r_{12}, Add_{11}, Add_{12}\rangle$ over elements of $ER(1) \cup R_{add,1}$ and $\langle r_{21}, r'_{21}, r'_{22}, Add_{21}\rangle$ over elements of $ER(2) \cup R_{add,2}$. Thus, our vectors are:

$$\mathbf{C}_{i-1,1} = \begin{pmatrix} 2 & 1 & 2 \end{pmatrix} \qquad \mathbf{C}_{i-1,2} = \begin{pmatrix} 1 & 0 \end{pmatrix}$$

Step 1

For region 1, object a belong to Category 1, object # and special object e belong to Category 2 while no objects belong to Category 3. On the other hand, objects # and a in region 2 belong to Category 1 and Category 3, resp.

Step 2

The additional identity rules per region are given below.

$$R_{add,1} = \{Add_{11} : \# \to \#, Add_{12} : e \to e\}$$
$$R_{add,2} = \{Add_{21} : a \to a\}$$

Since only object # is in category 2, $List_{cat_2}$ for region 1 is composed of only a single element #.

Step 3 and 4

The trigger matrix for both region 1 and 2 are shown below

$$TM_{\Pi_{ECPe},1} = \begin{pmatrix} 1\ 0\ 0 \\ 1\ 0\ 0 \\ 1\ 0\ 1 \\ 0\ 1\ 1 \\ 0\ 1\ 0 \\ 0\ 0\ 1 \end{pmatrix} \qquad TM_{\Pi_{ECPe},2} = \begin{pmatrix} 0\ 1 \\ 1\ 0 \end{pmatrix}$$

The extended application vectors $\mathbf{a'}_{i,1}$ and $\mathbf{a'}_{i,2}$ representing the vector of unknowns has the same index as that of the rows of their corresponding effect matrix.

Step 5

The resulting system of linear equations achieved from Equation (2) for region 1 and 2 is given below:

$$\begin{gathered} \mathbf{a'}_{i-1,1}(r_{11}) + \mathbf{a'}_{i-1,1}(r_{12}) + \mathbf{a'}_{i-1,1}(r'_{21}) = 2 \\ \mathbf{a'}_{i-1,1}(r'_{22}) + \mathbf{a'}_{i-1,1}(Add_{11}) = 1 \\ \mathbf{a'}_{i-1,1}(r'_{21}) + \mathbf{a'}_{i-1,1}(r'_{22}) + \mathbf{a'}_{i-1,1}(Add_{12}) = 2 \\ \mathbf{a'}_{i-1,2}(r_{21}) = 0 \\ \mathbf{a'}_{i-1,2}(Add_{21}) = 1 \end{gathered}$$

As can be traced, there are 11 possible extended application vectors for region 1 while there exists a unique extended application vector for region 2. Shown below is extended application vector for region 2:

$$\mathbf{a'}_{i-1,2} = \begin{pmatrix} 0\ 1 \end{pmatrix}$$

Step 6

The additional inequality in region 1 requires that:

$$\mathbf{a'}_{i,1}(Add_{12}) < 1$$

for cases where the trap object # remain. Thus, these solutions are possible:

$$\mathbf{a}_{i,1} = \begin{pmatrix} 0\ 0\ 2\ 0\ 1\ 0 \end{pmatrix} \qquad \mathbf{a}_{i,1} = \begin{pmatrix} 1\ 0\ 1\ 1\ 0\ 0 \end{pmatrix}$$

$$\mathbf{a}_{i,1} = \begin{pmatrix} 0\ 1\ 1\ 1\ 0\ 0 \end{pmatrix}$$

For cases where the trap object does not remain, the following solutions are also possible:

$$\mathbf{a}_{i,1} = \begin{pmatrix} 1\ 1\ 0\ 1\ 0\ 1 \end{pmatrix} \qquad \mathbf{a}_{i,1} = \begin{pmatrix} 0\ 2\ 0\ 1\ 0\ 1 \end{pmatrix}$$

$$\mathbf{a}_{i,1} = \begin{pmatrix} 2\ 0\ 0\ 1\ 0\ 1 \end{pmatrix}$$

After step 6, the 11 solutions in step 5 were reduced to six.

Step 7

Performing Equation (4) and Equation (5), below are the possible application vector combinations:

$$\begin{aligned}
&\textit{Solution } 1: && \mathbf{a}_{i,1} = \begin{pmatrix} 0\ 0\ 2\ 0 \end{pmatrix} && \mathbf{a}_{i,2} = \begin{pmatrix} 0\ 2\ 0 \end{pmatrix} \\
&\textit{Solution } 2: && \mathbf{a}_{i,1} = \begin{pmatrix} 1\ 0\ 1\ 1 \end{pmatrix} && \mathbf{a}_{i,2} = \begin{pmatrix} 0\ 1\ 1 \end{pmatrix} \\
&\textit{Solution } 3: && \mathbf{a}_{i,1} = \begin{pmatrix} 0\ 1\ 1\ 1 \end{pmatrix} && \mathbf{a}_{i,2} = \begin{pmatrix} 0\ 1\ 1 \end{pmatrix} \\
&\textit{Solution } 4: && \mathbf{a}_{i,1} = \begin{pmatrix} 1\ 1\ 0\ 1 \end{pmatrix} && \mathbf{a}_{i,2} = \begin{pmatrix} 0\ 0\ 1 \end{pmatrix} \\
&\textit{Solution } 5: && \mathbf{a}_{i,1} = \begin{pmatrix} 0\ 2\ 0\ 1 \end{pmatrix} && \mathbf{a}_{i,2} = \begin{pmatrix} 0\ 0\ 1 \end{pmatrix} \\
&\textit{Solution } 6: && \mathbf{a}_{i,1} = \begin{pmatrix} 2\ 0\ 0\ 1 \end{pmatrix} && \mathbf{a}_{i,2} = \begin{pmatrix} 0\ 0\ 1 \end{pmatrix}
\end{aligned}$$

The corresponding configuration vectors for each solution is as follows:

$$\begin{aligned}
&\textit{Solution } 1: && \mathbf{C}_{i,1} = \begin{pmatrix} 0\ 1\ 0 \end{pmatrix} && \mathbf{C}_{i,2} = \begin{pmatrix} 3\ 0 \end{pmatrix} \\
&\textit{Solution } 2: && \mathbf{C}_{i,1} = \begin{pmatrix} 2\ 0\ 0 \end{pmatrix} && \mathbf{C}_{i,2} = \begin{pmatrix} 2\ 1 \end{pmatrix} \\
&\textit{Solution } 3: && \mathbf{C}_{i,1} = \begin{pmatrix} 0\ 0\ 2 \end{pmatrix} && \mathbf{C}_{i,2} = \begin{pmatrix} 2\ 1 \end{pmatrix} \\
&\textit{Solution } 4: && \mathbf{C}_{i,1} = \begin{pmatrix} 2\ 0\ 3 \end{pmatrix} && \mathbf{C}_{i,2} = \begin{pmatrix} 1\ 1 \end{pmatrix} \\
&\textit{Solution } 5: && \mathbf{C}_{i,1} = \begin{pmatrix} 0\ 0\ 5 \end{pmatrix} && \mathbf{C}_{i,2} = \begin{pmatrix} 1\ 1 \end{pmatrix} \\
&\textit{Solution } 6: && \mathbf{C}_{i,1} = \begin{pmatrix} 4\ 0\ 1 \end{pmatrix} && \mathbf{C}_{i,2} = \begin{pmatrix} 1\ 1 \end{pmatrix}
\end{aligned}$$

2.4 A Sequential Implementation of Computation on ECPe Systems without Antiport using Initial Matrix Representation

Given the representation and algorithm for forward computing presented in Section 2.3, we were able to do a sequential implementation of computation on ECPe systems without antiport using the C programming language.

The system starts with reading two input files containing information for finding valid application vectors and determining the next configuration vector given a currently examined configuration vector. The following are the names of the input files:

- File *trans_file.txt* which contains the information needed for transitioning from one configuration to the next.
- File *forwComp_file.txt* which contains the information needed to find valid configuration vector/s given a current configuration vector.

A discussion about the format for the specified files is given in the appendix. The files will be used to initialize the necessary variables and pointers needed for the simulation. The system has two output files representing the tree-structure of the configuration history. The following are the name of the output files:

- File *conf.txt* which contains a list of configuration.
- File *conf_index.txt* which contains the index of the configuration in the tree structure.

Presented in Figure 2 is the flowchart of how the program works. Upon reading input files and loading variables and pointers needed for computing, the initial configuration is placed in the *conf.txt* and the associated index 1 is placed in *conf_index.txt*. Afterwards, the system will enter a loop for determining the configurations generated by a currently examined configuration. The examination of configuration will be executed in order of their position in the file. Given two configuration C and C' where the position of C precedes C', then configuration C will be examined first before configuration C'. Examining configuration shall halt only when the system reaches two stopping criterion:

- Upon achieving a pre-specified upper bound on the number of iterations
- Upon reaching a state where there are no more configurations to examine.

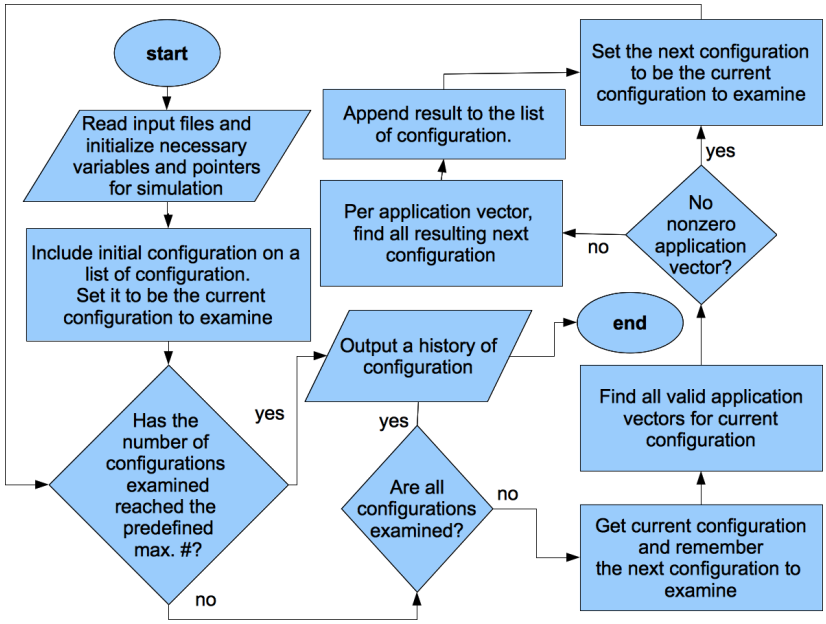

Fig. 2. An overview of our sequential implementation for ECPe systems without antiport in the C programming language.

For every loop, a configuration in the file is examined by first determining all valid application vectors which is applicable to the currently examined configuration. In finding a valid application vector, we use the concept of localized representation and extended application vectors, and follow the steps in Section 2.3 for forward

computing. If the only application vector applicable is the zero vector which means no more rule combination can be applied to the current configuration, it will proceed to the next configuration to examine. Each non-zero application vector is used to generate the next set of configurations. Afterwards, output files *conf.txt* and *conf_index.txt* will be updated to account all the newly generated next configuration vectors. Note that in this system, we have not yet implemented the methodology to detect repeating configuration.

On Generating Extended Valid Application Vectors and Filtering

The goal of step 5 in Section 2.3 is the generation of all possible extended application vectors $a'_{i,h}$ satisfying equation (2). In our sequential implementation, we achieve this by examining each equation resulting from the corresponding and equivalent system of linear equation. We now study the characteristics of the resulting system by using the example presented for forward computing.

Shown below are the equations yield from Equation (2) of region 1 for the example in Section 2.2, also shown in Section 2.3.

$$\beta_1 : \mathbf{a'}_{i-1,1}(r_{11}) + \mathbf{a'}_{i-1,1}(r_{12}) + \mathbf{a'}_{i-1,1}(r'_{21}) = 2$$
$$\beta_2 : \mathbf{a'}_{i-1,1}(r'_{22}) + \mathbf{a'}_{i-1,1}(Add_{11}) = 1$$
$$\beta_3 : \mathbf{a'}_{i-1,1}(r'_{21}) + \mathbf{a'}_{i-1,1}(r'_{22}) + \mathbf{a'}_{i-1,1}(Add_{12}) = 2$$

It can be observed that each equation in the resulting system represents an *object condition*; the object referring to possible objects that may enter an examined region. Also, for sending regions, an equation for *energy condition* (β_3) must also be present in a resulting system. In the general case, each variable (representing rule application of a certain communication rule) in the energy equation is present in exactly one other object condition. This object is the communication trigger that will be communicated upon activation of the rule represented by the said variable. For example, the variable $\mathbf{a'}_{i-1,1}(r'_{21})$ is present in both β_1 and β_3. The same goes for variable $\mathbf{a'}_{i-1,1}(r'_{22})$ which is present in both β_2 and β_3.

Other than such type of variables, no more variables can be present in more than one equation. Moreover, while the coefficients of the terms in the energy equation can contain any positive integer, the coefficients of the terms for non-energy condition will always be one (due to the restriction of noncooperative rule format). Moreover, the set of rules $r \in TR(h)$ are all represented by the union of all rules (variables) represented in the non-energy conditions without the identity rules.

Given such observation, possible vectors $a'_{i,h}$ are determined by first solving the condition posed for energy. Since there can be multiple solution for rules involving energy (the rules include the identity rule for energy since it is of Category 2), we shall determine the possible extended applications vectors resulting from a valid energy solution. In the linear equation shown above, we first on determining solutions for β_3. The possible *energy solutions* are

$$(1,1,0),(1,0,1),(0,1,1),(2,0,0),(0,2,0),(0,0,2),$$

where in each vector, the first element corresponds to value of $\mathbf{a'}_{i-1,1}(r'_{21})$, the second corresponds to the value of $\mathbf{a'}_{i-1,1}(r'_{22})$ and the third element is for the energy identity rule $\mathbf{a'}_{i-1,1}(Add_{12})$. For each solution, we then copy the rule application to the associated object to communicate. Afterwards, the rule application is transferred to the right-hand side of the equation, i.e. subtracted from the current count of the corresponding communicated object. As an example, the resulting modified object condition β'_1 caused by the value of communication rule$\mathbf{a'}_{i-1,1}(r'_{21})$ in the first energy solution will be

$$\beta'_1 : \mathbf{a'}_{i-1,1}(r_{11}) + \mathbf{a'}_{i-1,1}(r_{12}) = 1$$

If the resulting count is negative, then, the resulting energy solution cannot be applied. Therefore, no vector $a'_{i,h}$ can be generated given such negative result. This filtering on energy solution is evident in applying energy solution (0,2,0) on β_2. Upon subtracting the resulting rule application from the right-hand side of the corresponding communicated object condition, the resulting object conditions can be analyzed one at a time.

Upon realizing solutions for each object condition, step 6 of the forward computing methodology can already be executed per object condition. Identity rules for category 2 objects can be further checked to execute the filtering part, done in Step 6 of the methodology in Section 2.3 to see if the number of category 2 objects remaining in the region can be allowed to remain (that is, the case doesn't validate the rule that all objects that can evolve or be communicated must do so). Otherwise, the object solution will be dropped. Note that while step 6 in Section 2.3 evaluates first all extended application vectors before this step, we perform this step per object condition since the identity rules for any category 2 objects can only be present in the corresponding category 2 object equation. Preferably, the order of analyzing category 2 objects follow the sorted list $List_{cat_2}$ so that if an object solution is not valid, it can terminate immediately at the first unsatisfied category 2 object.

The resulting value of each filtered variable set per solution can be combined, one solution from each object, and each combined list constitutes one extended application vector. To illustrate this, we examine the possible extended application vectors that can be yield from energy solution $(1, 1, 0)$. For β_1 condition, the object solutions are $(0, 1)$ and $(1, 0)$ where the first element of the said vectors correspond to $\mathbf{a'}_{i-1,1}(r_{11})$ and the second, to $\mathbf{a'}_{i-1,1}(r_{12})$. For β_2 condition, the object solution only assigns the value 1 to $\mathbf{a'}_{i-1,1}(Add_{11})$. Therefore, for energy solution (1,1,0), the corresponding extended application vectors yielded are

$$\mathbf{a}_{i,1} = \begin{pmatrix} 1\,0\,1\,1\,0\,0 \end{pmatrix} \qquad \mathbf{a}_{i,1} = \begin{pmatrix} 0\,1\,1\,1\,0\,0 \end{pmatrix}$$

3 Simulator design and implementation

In this section, we relay how we can employ GPUs to parallelize the task of finding all possible object solutions. NVIDIA introduced the Compute Unified Device

Architecture (CUDA) in 2007 [7]. CUDA is a software and hardware architecture for general purpose computations in NVIDIA's GPUs [7]. CUDA extends high-level languages such as *C* to allow programmers to easily create software that will be executed in parallel, avoiding low-level graphics and hardware primitives [12].

GPUs introduce increased performance speedups over CPU only implementations with linear algebra computations (among other types of computations) because of the GPU architecture. The common CPU architectures are composed of transistors which are divided into different blocks to perform the basic tasks of CPUs (general computation): control, caching, DRAM, and ALU (arithmetic and logic). In contrast, only a fraction of the CPU's transistors allocated for control and caching are used by GPUs, since far more transistors are used for ALU [7] (see Figure 3 for an illustration). This architectural difference is a very distinct and significant reason why GPUs offer large performance increases over CPU only implementation of parallel code working on large amounts of input data. However if the problem to be solved cannot be organized in a data parallel form (a task performing computations on data need not depend heavily on other task's results) then the performance of GPUs over CPUs will not be fully utilized.

Code written for CUDA can be split up into multiple threads within multiple thread blocks, each contained within a grid of (thread) blocks. These grids belong to a single device or GPU. Each device has multiple cores, each capable of running its own threads. Each core in the device is able to run a set of threads.A thread block is assigned to each multiprocessor, where each processor is made up of several cores [7, 12]. A function known as a *kernel function* is one that is called from the host or CPU but executed in the device. Using kernel functions, the programmer can specify the GPU resources: the layout of the threads (from one to three dimensions) in a thread block, and the thread blocks (from one to two dimensions) in a grid. Table 1 shows the resources of current CUDA enabled NVIDIA GPUs.

GPU resources	**Values**
Global memory	Up to 4GB
Max number of threads per dimension (x, y, z)	$(1024, 1024, 64)$
Max number of thread blocks per grid (x, y, z)	$(65535, 65535, 65535)$

Table 1. Typical resources for CUDA enabled Fermi architecture GPUs (from [7, 12]) .

On Parallelizing Transitions

Another apparent possibility in order to simulate the parallel computations of ECPe systems (as well as capitalize on their representations as matrices) on GPUs, we can have initially at least two levels of parallelism: the first level is the computation of Equation (1) in parallel by threads in a block; the second level involves the computation of all the possible next configurations given a current configuration, so that each block in a grid of thread blocks performs this level.

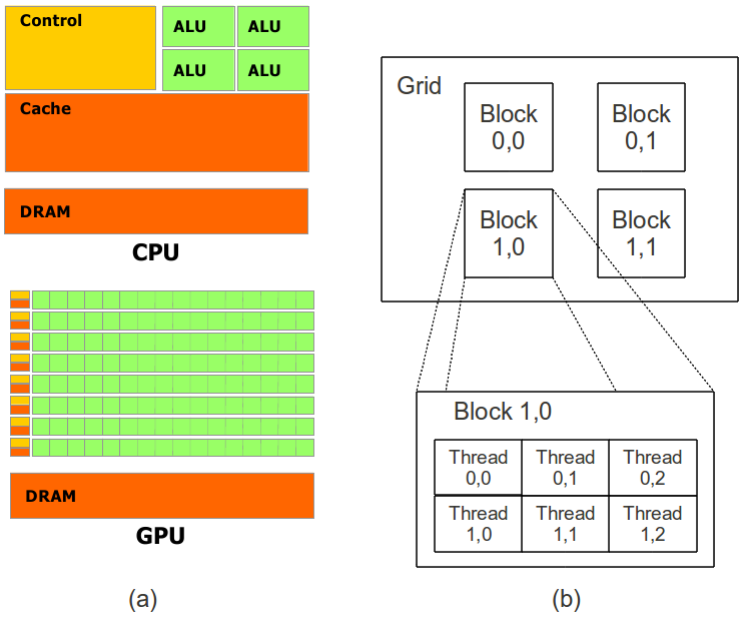

Fig. 3. (a) Common transistor allocation of CPUs and GPUs (b) Computing unit hierarchy of GPUs, from [12].

The first level is highly parallelizable since vector-matrix multiplication and vector addition are highly data independent. Each thread can multiply a vector to one column of the matrix, thus performing $a_{i,h} \cdot M_{\Pi,h}$. Each thread sums the products then adds these to another vector, performing the addition of $C_{i-1,h}$. For the second level, if there are q number of $a_{i,h}$'s and hence q number of next configurations, then q blocks will perform Equation (1) q times.

Because of the physical limitations of current NVIDIA GPUs, no more than 1024 threads per block are allowed for Fermi architecture GPUs so that at most matrices of at most 1024 columns can be simulated in a block. In Fermi GPUs, the maximum number of allowable thread blocks in a grid is 65535 (See Table 1) so q is currently upper bounded by this value. Another simulation consideration, aside from the computing units (threads, thread blocks) is the relatively more limited memory of current GPUs compared to CPUs. In this case, storing all q number of application vectors (each of which are of length $|R(h)|$) and the q number of next configurations (each of which are of length $|PO(h)|$) resulting from those application vectors must fit into the GPU's global memory.

On Generating Object Solutions

Given an examined energy solution, we check each object condition where each variable corresponding to a communication rule have already been determined (via

the examined energy solution) and the value at the right-hand side of a communicated object has already been updated. As can observed, this problem is reduced to an integer partition problem where, given an object equation $\alpha_1+\alpha_2+...+\alpha_k = n$, we need to find vector containing $\alpha_i \in \mathbb{N}$'s, i.e. $(\alpha_1, \alpha_2, ..., \alpha_k)$.

To generate solutions for non-energy object equation, we first obtain a *lexicographic order* of partition for the value n. In [13], given $X = (x_1, x_2, ..., x_{k'})$ and $Y = (y_1, y_2, ..., y_{k''})$, X precedes Y lexicographically if and only if for some $j \geq 1$, $x_i \geq y_i$ when $i < j$, and x_j precedes y_j. As an example, partitions of 5 in lexicographic order are: $11111, 2111, 311, 221, 311, 32, 41, 5$.

Each resulting partition will be padded with zeroes accordingly so that the partition can be represented in a k-dimensional vector. Each partition will be assigned to a thread. Each thread will be responsible for the generation of distinct permutation of the k-dimensional vector representing the partition. The union of solutions generated by each partition corresponds to the set of object solutions for a certain examined object. Since the filtering step of the forward methodology, as explained in Section 2.4, can be done per object condition, this step can also be performed within the current threads.

The idea of parallelization in Section 3 may also be used for generating energy solutions of the form $c_1\alpha_1 + c_2\alpha_2 + ... + c_k\alpha_k = n$, where a lexicographic order on the partitions of n is first accomplished. Again, each resulting partition will then be padded with zeroes accordingly. Upon assigning each partition to a thread, and generating distinct permutation of a partition, each vector representating a permutation can be equated with the vector $(c_1\alpha_1, c_2\alpha_2, ..., c_k\alpha_k)$, afterwhich the corresponding value for the $\alpha_i's$ can be obtained.

4 Conclusions and future work

In this report, we were able to describe a sequential implementation of a forward computing methodology for ECPe systems without antiport rules. We also were able to show how we extend this sequential work to employ GPUs for parallelizing some key areas in the implementation procedures. We were also able to show our proposed ideas for parallelizing other parts of the code.

As future work, we would like to implement our proposed ideas for parallelization and test the efficiency of the resulting implementation. Moreover, we hope to improve these ideas to better capitalize the parallelizability of vector-matrix representations of the computations on GPUs. As part of our future works, we also would like to extend the methodology for forward computing to apply to a general ECPe systems where antiport rules are allowed. The difficulty in allowing such communication rules are influenced by the implication that a region can be both a sender and receiver region. Thus, antiport rules need to maintain relationship between adjacent regions at crucial parts of the forward computing methodology. The more tricky part is the action done when considering objects that are unmoved due to antiport rule. The antiport rule increases the number of possible cases dictating why a category 2 object can remain in a certain region.

5 Acknowledgments

R.B. Juayong and F.G.C. Cabarle are partially supported by the DOST-ERDT scholarship program. H. Adorna is funded by the DOST-ERDT research grant and the Alexan professorial chair of the UP Diliman Department of Computer Science, University of the Philippines Diliman. M.A. Martínez–del–Amor is supported by "Proyecto de Excelencia con Investigador de Reconocida Valía" of the "Junta de Andalucía" under grant P08-TIC04200, and by the project TIN2009–13192 of the "Ministerio de Ciencia e Innovación" of Spain, both co-financed by FEDER funds.

References

1. H. Adorna, Gh. Păun, M. Pérez-Jiménez : On Communication Complexity in Evolution-Communication P systems, *Romanian Journal of Information Science and Technology, Vol. 13 No. 2 pp. 113-130, 2010*
2. F.G.C. Cabarle, H. Adorna, M.A. Martínez-del-Amor: A Spiking Neural P system simulator based on CUDA, M. Gheorghe et al. (Eds.), *12th Int'l Conference on Membrane Computing 2011*, revised and selected papers, LNCS vol. 7184, pp. 87-103. Springer-Verlag, 2012.
3. F.G.C. Cabarle, H. Adorna, M.A. Martínez-del-Amor, M.J. Pérez-Jiménez: Improving GPU Simulations of Spiking Neural P Systems, (to appear) *Romanian Journal of Information Science and Technology*, 2012.
4. R.A. Juayong, H. Adorna: Computing on Evolution-Communication P systems with Energy Using Symport Only, *Workshop on Computation: Theory and Practice 2011 (WCTP 2011), UP Diliman NISMED auditorium.*
5. X. Zeng, H. Adorna, M. A. Martínez-del-Amor, L. Pan, M. Pérez-Jiménez : Matrix Representation of Spiking Neural P Systems, *Membrane Computing: Lecture Notes in Computer Science, Volume 6501/2011, 377-391, 2011*
6. R.A. Juayong, H. Adorna: A Matrix Representation for Computations in Evolution-Communication P Systems with Energy, *Proc. of Philippine Computing Science Congress, Naga, Camarines Sur, Philippines, March 3-4, 2011*
7. Kirk D., Hwu W., *Programming Massively Parallel Processors: A Hands On Approach*, 1st ed. MA, USA, Morgan Kaufmann, 2010.
8. M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez: Computing Backwards with P systems, $WMC10$, Curtea de Argeş, Romania, (2009), 282-295.
9. Gh. Păun: Introduction to Membrane Computing. In: Gabriel Ciobanu, Mario J. Pérez-Jiménez and Gheorghe Păun, eds: Applications of Membrane Computing, Natural Computing Series. Springer, pp.142. (2006)
10. M. Cavaliere: Evolution-communication P systems. *Membrane Computing. Proc. WMC 2002, Curtea de Argeş* (Gh. Păun et al., eds.), LNCS 2597, Springer, Berlin, 134-145. (2003)
11. A. Păun, Gh. Păun: The power of communication: P systems with symport/antiport. *New Generation Computing, 20, 3, 295-306*, (2002)
12. NVIDIA corporation, *"NVIDIA CUDA C programming guide"*, version 3.2, CA, USA, 2010.
13. A. Zoghbi, I. Stojmenović: Fast algorithms for generating integer partitions, *International Journal of Computer Mathematics*, Vol. 70, No. 2., pp. 319-332, (1998)

Appendix A:
On File Formats for Implementation of ECPe system in C

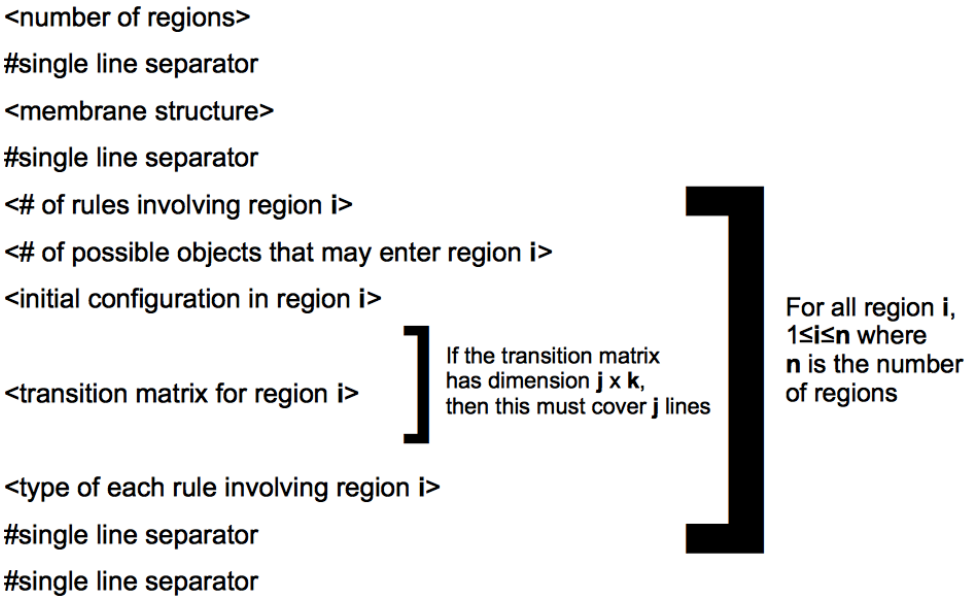

Fig. 4. Format for file *trans_file.txt* for the sequential implementation of ECPe system in C.

Shown in Figure 4 above is the format for input file *trans_file.txt*. This file contains the information required to find a succeeding configuration given a current one. First, the number of regions must be specified. If there will be a case where the environment will be needed during the computation, the value of this parameter must be the number of membranes incremented by one in order to account for the environment. The membrane structure is represented by paired square brackets as typical representation of a membrane structure. Note that for either a closed or open square bracket, the symbol must be followed by a numeral to indicate the label of the membrane. For the initial configuration, there is a need to follow a total order as discussed in Section 2.3. Therefore, the initial configuration is simply a initial configuration vector where each cell contains the number of copies of a certain object at the start of the computation. The separator for the cells will be the space symbol. The expected transition matrix has dimensions following the previously specified number of rules involving a specific region and the number of possible objects that may enter the region. The rows will be separated by newline, whereas, the columns are separated by spaces. A rule type can either be an evolution rule in

which case the symbol to type will be 'e', whereas a communication rule can have one of symbol 's' and 'r'. The symbol 's' represents that the communication rule uses the specific region as a sender, while the symbol 'r' uses the specific region as a receiver.

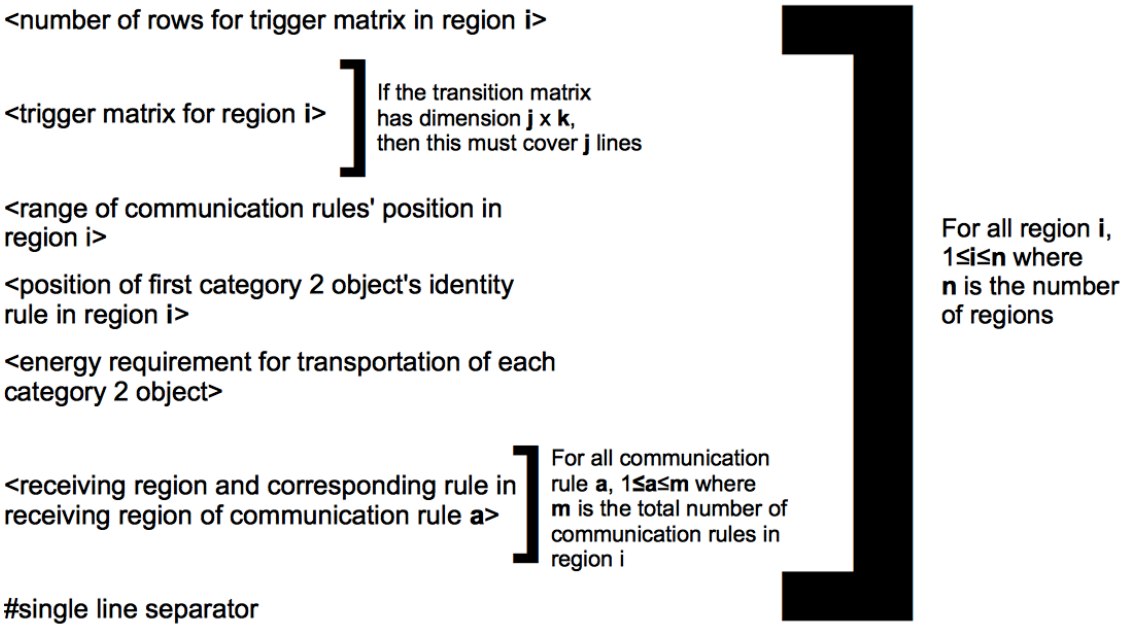

Fig. 5. Format for file *forwComp_file.txt* for the sequential implementation of ECPe system in C.

Figure 5, on the other hand, illustrates the format for the input file *forwComp_file.txt*. This file contains the information required to find all valid application vectors from a given configuration vector. First, a trigger matrix needs to be indicated. Again, there is a need to impose a total order over the rules. In this case, there will be a specific setup for the rules in the trigger matrix wherein, it is required that evolution rules must be specified first before the communication rules. After the existing rules associated with the region, the identity rule for energy (e) should proceed after. The identity rules for the other category 2 objects will follow after the energy's identity rule. After indicating the trigger matrix, there is a need to define the range of the communication rules in the specified region. To indicate this:

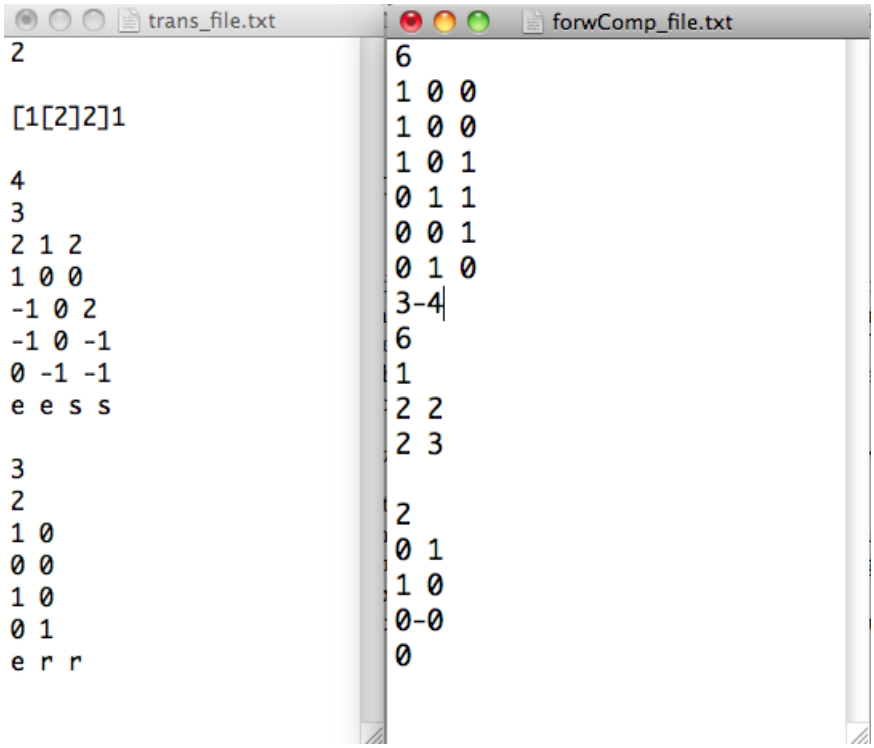

Fig. 6. Input files for the example given in Section 2.2

<position of starting communication rule>-<position of last communication rule>

where the positions will be based on the imposed total order. The total order will also be used in the next set of input which entails specifying the minimal energy requirement needed to transport each category 2 object. Since they only be enumerated in a single line, the separator of energy per object will be the space symbol. Following after this will be the enumeration of the details needed in the receiving region for each communication rule. To indicate this:

<position of receiving region> <position of partner rule in receiving region>

where this specifications per communication rule will be separated by newlines. Note that only a single space symbol separates the position of the receiving region with the position of the corresponding rule in the receiving region. In case no communication rules exist in the trigger matrix, there is no need to fill up this part. Moreover, the range of communication rules will be 0-0 and the value of the succeeding line will be 0. Shown in Figure 6 are the input files for the example given in Section 2.2 whose total order for involved rules and possible objects follows the order given in Section 2.3 except that we swap the position of the last and second

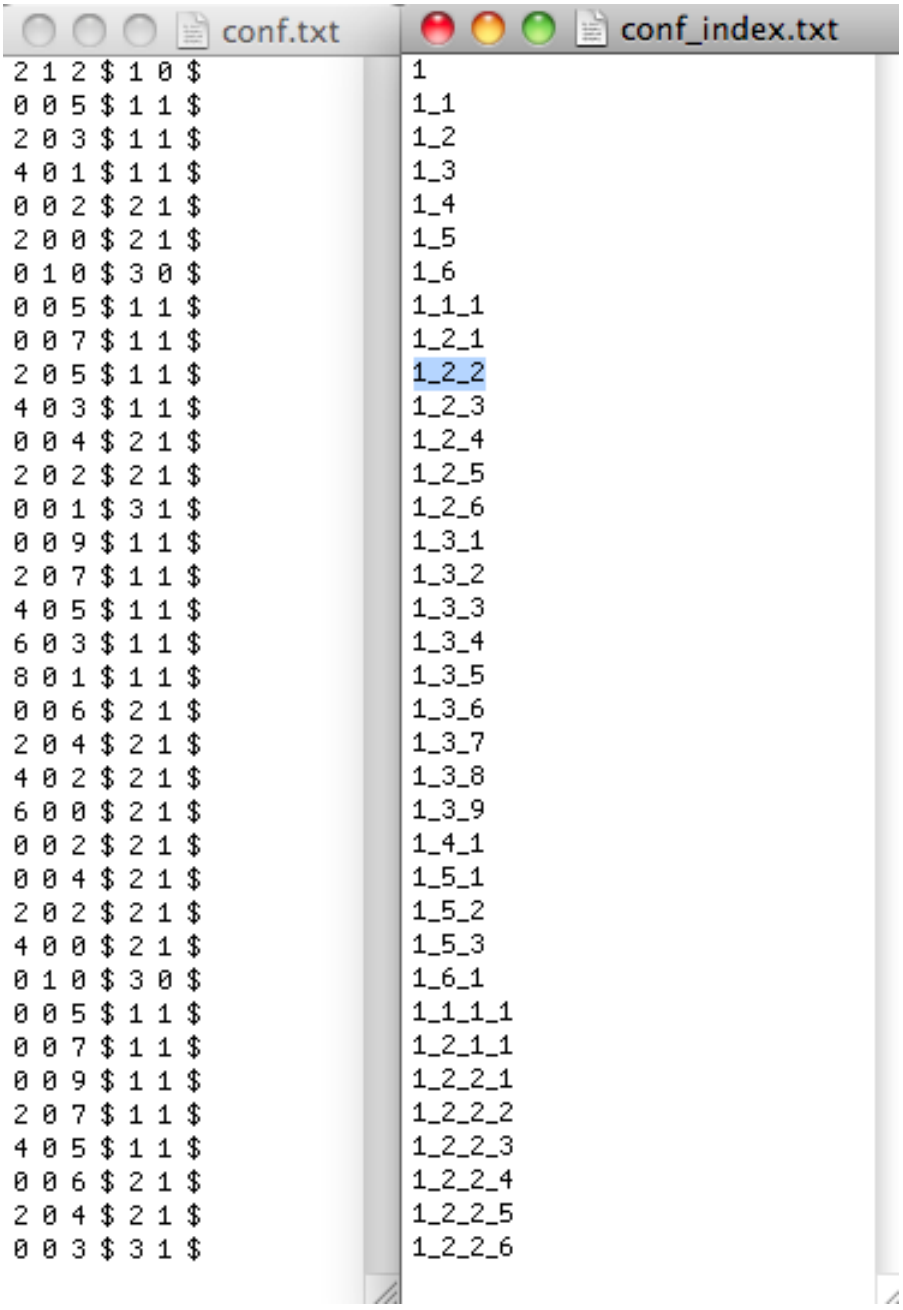

Fig. 7. Sample output files for the example given in Section 2.2

to the last identity rule in the first region to follow the required format for total order on rules involved in the corresponding trigger matrix.

For the output, there will be two files, namely *conf.txt* and *conf_index.txt*, that shall represent an ECPe systems configuration tree of computations. The former consists a list of configurations (not yet necesarily unique) where each system configuration is separated by a newline. The system configuration is composed of configuration vectors local to each region that are juxtaposed together. The elements of the vectors are separated by an individual space whereas dollar sign ($) separates each local configurations. The initial configuration will be the first configuration in the file.

The output file *conf_index.txt* stores the indices of the configurations in file *conf.txt* in order to remember the association between configurations. Given an index in this file, the configuration associated with the index is the configuration in *conf.txt* which has the same line position as the line of the index. For example, the initial configuration is located at the first line in the file $conf.txt$, its corresponding first line in $conf\_index.txt$ is the index 1. Since we can represent computation as a tree, the initial configuration with index 1 is the root node configuration of the tree. To determine the configurations that sprung from the initial configuration i.e. the children of the root node, the indices must have the prefix '1_'. The parent configuration which generates a configuration can be tracked by removing the last underscore in the index associated with the configuration, along with the number that appears after the said underscore. A path from the initial configuration to any configuration C can be traced by retrieving the associated configuration starting from index 1 (which, as mentioned, represents the initial configuration) to the configuration C following the precedence imposed by the indices. Shown in Figure 7 are the output files for the example given in Section 2.2 whose total order for objects follows the order given in Section 2.3 and where the maximum number of configurations to examine is set to 10.