A GPU Simulation for Evolution-Communication P Systems with Energy Having no Antiport Rules

Zylynn F. Bangalan[1], Krizia Ann N. Soriano[1], 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: `zfbangalan@up.edu.ph`, `knsoriano1@up.edu.ph`, `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. Evolution-Communication P system with energy (ECPe systems) is a cell-like variant P system which establishes a dependence between evolution and communication through special objects, called 'energy,' produced during evolution and utilized during communication. This paper presents our initial progress and efforts on the implementation and simulation of ECPe systems using Graphics Processing Units (GPUs). Our implementation uses matrix representation and operations presented in a previous work. Specifically, an implementation of computations on ECPe systems without antiport rules is discussed.

1 Introduction

Evolution-Communication P system with energy (ECPe systems) is a modification to Evolution-Communication P system (ECP system). Objects evolve through *evolution rules* while communication of objects to other regions bounded by membranes is done through *symport/antiport rules*. The unique features of an ECPe system not present for Evolution-Communication P system are listed below.

- A communication rule must require at least one quantum of energy, referred to as e, to be triggered.
- This quantum of energy, e, is produced by evolution rules and consumed by communication rules only i.e. it can never be present at the initial configuration.

ECPe systems are actually presented to provide a measure for communication over Evolution-Communication P systems [1].

Other important characteristics of ECPe systems which are common to other variants of P systems are also worth mentioning. Objects in each membrane are considered as multisets since there can be multiple instances and type of objects present within membranes. Evolution rules and communication rules within each membrane are applied in a *nondeterministic maximally parallel manner*. Maximal parallelism ensures that all rules that can be applied must be applied given the multiset of objects in each membrane while nondeterministic application of rules arises because it is possible that more than one rule is applicable at the same time. Further definition and discussion about ECPe systems is found in section 2.

Computations in ECPe system have been represented using vectors, matrices and linear algebra in [6]. As suggested in [8], we can make these vectors and matrix representations local to regions to avoid dealing with sparse vectors and matrices and to make computations in the system suitable for parallel processing.

Many works have been made for simulating P systems. Efforts for simulation is motivated by the fact that simulations help in the analysis of P systems. Since P systems are highly parallel in nature, many of the works are focused on its parallel implementation. One example is the parallel implementation of Transition P systems on a cluster of computers presented in [5]. Another is that in [3] wherein Spiking Neural P systems are simulated in parallel using GPUs. Though there are already a number of parallel implementations for P systems, none of these is directly applicable to ECPe systems. We are interested in a parallel implementation of ECPe systems in GPUs to contribute on researches implementing cell-like variant of P systems on GPUs. We also aim to spark interest and aid in further researchers in the field of Membrane computing, particularly ECPe systems, and parallel computing paradigms in general.

Just like the work in [3], we will make use of NVIDIA GPUs for this parallel simulation and implementation. NVIDIA, a manufacturer of GPUs, introduced *Compute Unified Device Architecture* (CUDA). CUDA is a software and hardware architecture that permits programmers to have a control over NVIDIA's GPU hardware and use them for parallel computations.

Our parallel implementation of ECPe systems is a continuation on an earlier study [8]. Algorithms for the methodology presented in [8] are developed and are implemented in CUDA C. It is important to emphasize that the implementation done only involves methodology for forward computing of Evolution Communication P system without antiport rules only

2 ECPe system

2.1 Definitions and notations

ECPe system is introduced in [1] as a model where *special objects* are used to establish dependence of communication on evolution. The goal of [1] is mainly to initiate communication complexity analysis for P systems. In order to evaluate *communication* complexity for computations in ECPe systems, a cost of using a communication rule is considered. This cost is in the form of a quantity of "energy". A single object can be transported by a communication rule with the help of one or more quantum of energy, e. This quantum of energy is a special object, $e \notin O$, which can be produced by evolution rules and consumed through communication rules only. No communication rules can be applied without consuming an amount of "energy". When objects are transported, the quanta of energy consumed are lost. They do not pass across membranes.

Following the definition in [1], an EC P System with energy is a construct of the form

$$\Pi = (O, e, \mu, \omega_1, ..., \omega_m, R_1, R'_1, ..., R_m, R'_m, i_{out})$$

where:

1. O is the alphabet of objects;
2. m is the total number of membranes;
3. μ is the membrane structure
   Membrane i is called the *parent membrane* of a membrane j, denoted $parent(j)$, if the paired-labelled square brackets corresponding to membrane j is inside the paired-labelled square brackets corresponding to membrane i, i.e., $[_i \ldots [_j\ ]_j]_i$. Conversely, membrane j is called a *child membrane* of membrane i, denoted $j \in children(i)$ where $children(i)$ is referring to the set of membranes inside membrane i.
4. $\omega_1, ..., \omega_m$ are multisets of objects present in the regions bounded by membranes;
5. $R_1, ..., R_m$ are sets of evolution rules, each associated with a region delimited by a membrane;
   An evolution rule is of the form $a \rightarrow v$ where $a \in O$, $v \in (O \cup \{e\})^*$ i.e e should never be in the initial configuration and cannot be evolved.
6. $R'_1, ..., R'_m$ are sets of symport/antiport rules each associated with a membrane; A symport rule is of the form (ae^i, in) or (ae^i, out), where $a \in O$, $i \geq 1$. The number i is called the *energy* of the rule. An antiport rule is of the form $(ae^i, out; be^j, in)$ where $i, j \geq 1$. The number $i + j$ is called the *energy* of the rule.
7. $i_{out} \in \{0, 1, ..., m\}$ is the output region where i_0 is the environment;

As in classical cell-like variants, rules must be applied in a nondeterministic and maximally parallel manner. A configuration at any time i is denoted by C_i while 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 the system doesn't halt, this implication failure of computation because the system did not produce any output.

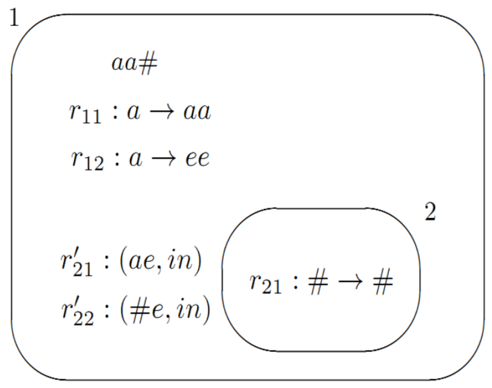


Fig. 1. An ECPe system with a construct of $\Pi = (\{a, \#, e\}, e, [_1[_2]_2]_1, \omega_1, \omega_2, R_1, R_2, R'_1, R'_2, 2)$ where $\omega_1 = \{a^2, \#\}$, $\omega_2 = \emptyset$, $R_1 = \{r_{11} : a \to aa, r_{12} : a \to ee\}$, $R_2 = \{\# \to \#\}$, $R'_1 = \emptyset$, $R'_2 = \{r'_{21}(ae, in), r'_{22} : (\#e, in)\}$

2.2 Matrix Representations

In [6], a matrix representation for ECPe system is obtained. With this representation, matrix operations can be used to model computations in ECPe system. The work is motivated by the fact that an algebraic representation helps in simulating P system. An implementation can help in easier or faster analysis of ECPe system. To obtain a matrix representation, a total order over the objects and over the rules are defined so that the elements can be identified by their positions. The following are also defined:

- Configuration vector $\mathbf{C}_i$ is a vector with length $|(O \cup \{e\}) \times \{1, ..., m\}|$ and whose elements are numbers representing the multiplicity of objects in each region at time i.
- Application vector $\mathbf{a}_i$ is a vector with length $|\bigcup_{1 \leq k \leq m} R_k \cup R'_k|$ and whose elements are the number of times each rule is applied during a transition $C_{i-1} \Rightarrow C_i$.
- Transition matrix M_Π is an n by r matrix ($n = |\bigcup_{1 \leq k \leq m} R_k \cup R'_k|$ and $r = |(O \cup \{e\}) \times \{1, ..., m\}|$) that shows the effect of the application of each rule. The matrix $M_\Pi(r, (\alpha, k))$ gives the number of consumed or produced object α in region k when the rule r is applied once. Elements with negative value represent the number of objects consumed or moved out of a given region while elements with positive value represent the number of objects that will be produced or moved in a given region. Elements with zero value represent the objects that are either not involved in the rule or involved but the total effect of their production and consumption is zero.
- Trigger matrix is an $r \times n$ matrix that represents all the needed objects in order to activate the rule. The elements are the number of objects required for a rule to be applied.

(Formal definitions of the vectors and matrices above are presented in [6]).

Forward computing is the process of finding all possible next configuration given an input configuration. Equation 1 shows that given a configuration vector for a certain time $\mathbf{C}_{k-1}$, transition matrix M_{Π} and an application vector for a transition $C_{k-1} \Rightarrow C_k$, the next configuration vector $\mathbf{C}_k$ can be computed:

$$(\mathbf{C}_k = \mathbf{C}_{k-1} + \mathbf{a}_k \cdot M_{\Pi}) \tag{1}$$

The equation above implies that in order to compute all next configurations for forward computing, there is a need to find all valid application vectors $\mathbf{a}_k$. [7] shows that this problem can be reduced into solving a system of linear equations.

2.3 Localization of Computations

In [7], the localization of rules in ECPe system are taken into consideration, dividing equation (1) into multiple equations (one for every local region) making them suitable for parallel processing. Localization also provides a hint to unreachability based on rules and initial multiset of objects. The following notations are defined over an ECPe system Π without antiport rules.

- Let $IO(r, k)$ be the set of objects in region k involved in a rule r.
- Let $TO(r, k)$ be the set of objects in region k that trigger a rule r.

The following definitions and theorems taken from [7].

Definition 1 ***Involved Rules in Region*** k

$$IR(k) = R_k \cup R'_k \cup (\bigcup_{k' \in children(k)} R'_{k'})$$

Definition 2 ***Possible Objects in Region*** k

$$PO(k) = \{\alpha | \alpha \textit{ appeared in } w_k\} \cup (\bigcup_{r \in IR(k)} IO(r, k))$$

Definition 3 ***Effect Rules in Region*** k

$$ER(k) = \{r | r \in R_k\} \cup (IR(k) - \{r' | r' \in IR(k) \textit{ and } TO(r', k) \neq \emptyset\})$$

Definition 4 ***Trigger Rules in Region*** k

$$TR(k) = \{r | TO(r, k) \neq \emptyset\}$$

Definition 5 ***Configuration Vector for each Region*** k
A configuration vector $\mathbf{C}_{i,k}$ *is a vector whose length is* $|PO(k)|$. *The vector* $\mathbf{C}_{i,k}(\alpha)$ *refers to the multiplicity of object* α *in region* k *at configuration* C_i.

Definition 6 ***Application Vector for each Region*** k
An application vector $\mathbf{a}_{i,k}$ *is a vector whose length is* $|IR(k)|$. *The vector* $\mathbf{a}_{i,k}(r)$ *refers to the number of application of rule* r *specifically in region* k *during the transition* $C_{i-1} \Rightarrow C_i$.

Definition 7 ***Transition Matrix for each Region*** k
A transition matrix $M_{\Pi,k}$ is a matrix whose dimension is $|IR(k)| \times |PO(k)|$. The matrix $M_{\Pi,k}(r, \alpha)$ returns the number of consumed or produced object α in region k upon single application of rule r. The consumed objects have negative values while the produced objects are positive. If object α in region k is not used in rule r, then its value is zero.

Theorem 1 *(from [7]) The effect of Equation (1) is the same as the effect of performing*

$$\mathbf{C}_{i,k} = \mathbf{C}_{i-1,k} + \mathbf{a}_{i,k} \cdot M_{\Pi,k} \tag{2}$$

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

Corollary 1 *(from [7]) The formula for computing backward is*

$$\mathbf{C}_{i-1,k} = \mathbf{C}_{i,k} - \mathbf{a}_{i,k} \cdot M_{\Pi,k} \tag{3}$$

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

The above definitions and theorems are used to make vectors and matrices local to regions to exploit independence between regions for parallel computations.

2.4 Methodology for Forward Computing

Given $\mathbf{C}_{i,k}$, we determine $\mathbf{C}_{i+1,k}$ by forward computing using the methodology presented in [8].
1. Categorize all possible objects in $PO(k)$ for all region k. First, categorize all $\alpha \in PO(k)$ for a certain region k. The categories are:

- Category 1: Evolution Trigger
  Object α is in this category if there exists $r \in R_k$ such that $TO(r, k) = \{\alpha\}$.
- Category 2: Communication Trigger Only
  Object α belongs in this category if there does not exist $r \in R_k$ such that $TO(r, k) = \{\alpha\}$ but there exists $r' \in IR(k)$ such that $\alpha \in TO(r', k)$.
- Category 3: Not a Trigger
  Object α is not in Category 1 and is not in Category 2.

2. Construct identity rules for objects in Category 2 and 3 for all region k. For each $\alpha \in PO(k)$ that belongs to one of Category 2 and Category 3, include an identity rule $\alpha \to \alpha$. Place all these rules in a set labelled $R_{add,k}$. Also, keep a list of $\alpha' \in PO(k) - \{e\}$ that fall under Category 2. Call this list as $List_{cat_2}$ and sort it in increasing order of transport energy requirement.
3. Construct Trigger Matrix $TM_{\Pi,k}$ for all region k The defined rules associated with the rows of $TM_{\Pi,k}$ must be the rules that lessen the multiplicity of objects in region k. These rules are represented in the set $TR(k)$. Again, let

the additional rules from $R_{add,k}$ be represented in the rows as well. The set of objects represented in the columns of $TM_{\Pi,k}$ remains $PO(k)$. $TM_{\Pi,k}$ has dimensions $|TR(k) \cup R_{add,k}| \times |PO(k)|$. $TM_{\Pi,k}(r, \alpha)$ gives the multiplicity of α in region k that is required to apply rule r once.
4.Set the length of the vector of unknowns (extended application vector) $\mathbf{a'}_{i,k}$ for all region k The length of $\mathbf{a'}_{i,k}$ is $|TR(k) \cup R_{add,k}|$.
5. Solve system of linear equation Find all solutions to the equation

$$\mathbf{a'}_{i,k} \cdot TM_{\Pi,k} = \mathbf{C}_{i-1,k} \tag{4}$$

Again, because the application vector's ($\mathbf{a'}_{i,k}$) elements represent the number of application of rules, it must not contain negative numbers i.e. the elements must always be natural numbers. The value $\mathbf{a'}_{i,k}(r)$ returns either the number of application of each rule $r \in TR(k)$ or how many object α is unevolved or unmoved (if $(r : \alpha \to \alpha) \in R_{add,k}$). $TR(k)$ and $R_{add,k}$ are disjoint sets.
6. Filter solutions in step 5 For each region k, 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. $\mathbf{a'}_{i,k}(\alpha_{cat_2,min} \to \alpha_{cat_2,min}) > 0$. Since $List_{cat_2}$ is sorted in increasing order of energy requirement for transport, $\alpha_{cat_2,min}$ has the least energy required for communication. Label its corresponding energy as $energy(\alpha_{cat_2,min})$. Filter solutions in step 5 by adding, for each region k with a non-empty $List_{cat_2}$, the inequality below:

$$\mathbf{a'}_{i,k}(e \to e) < energy(\alpha_{cat_2,min}) \tag{5}$$

7. For each solution in step 6, find $\mathbf{a}_{i,k}$ When values for $\mathbf{a'}_{i,k}$ in all region k are found, disregard all identity rules $r' \in R_{add,k}$. Fill the values of an application vector $\mathbf{a}_{i,k}$ through the equation

$$\mathbf{a}_{i,k}(r) = \mathbf{a'}_{i,k}(r), \ r \in R_k. \tag{6}$$

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

$$\mathbf{a}_{i,k'}(r) = \mathbf{a}_{i,k''}(r) = \mathbf{a'}_{i,k'}(r) \tag{7}$$

for all communication rule $r \in IR(k') \cap IR(k'')$ where region k' is the sending region.

Theorem 2 *(from [7]) All possible $\mathbf{a}_{i,k}$ generated through the above methodology leads to a valid $\mathbf{C}_{i,k}$ yielded from $\mathbf{C}_{i-1,k}$ in one computational step.*

3 Graphics Processing Unit (GPU)

3.1 On Using GPUs

Although GPUs are initially used for image processing, it is actually designed to handle computationally demanding applications. Thus, its use is extended to accommodate different applications. GPU has been widely used to work with highly

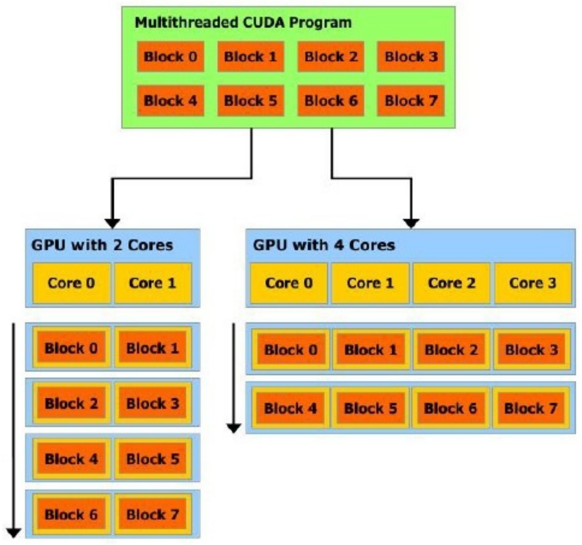


Fig. 2. NVIDIA CUDA automatic scaling(More cores, faster execution), from [3]

parallel applications due to its parallel nature as compared to setting-up multiple CPUs that will harness the same power by that of a GPU [3]. Another reason is that they provide not only the hardware but also application programming interfaces (APIs) for computation. As mentioned in [11], the GPU is designed to cater to a class of applications with the following characteristics,

- Computational requirements are large.
- Parallelism is substantial.
- Throughput is more important than latency.

3.2 Compute Unified Device Architecture (CUDA)

The Compute Unified Device Architecture (CUDA) programming model is introduced by NVIDIA, a manufacturer of GPUs. CUDA is a hardware and software architecture that runs highly parallel computations on the family of GPUs manufactured by NVIDIA [3]. With this feature and compatibility with today's leading GPU devices, CUDA became popular and progress has been made to make programming in CUDA easier. Though CUDA is an extension of the C programming language for parallel computations, programmers can also access CUDA APIs with FORTRAN, Haskell, Perl, Python, Ruby, and etc.

The parallel code written in extensions of the C programming language is executed in multiple threads within multiple blocks which are in turn parts of a grid of blocks. These blocks belong to the GPU. Each GPU consists of multiple cores having their own block of threads [3]. This feature is illustrated in Figure 2.

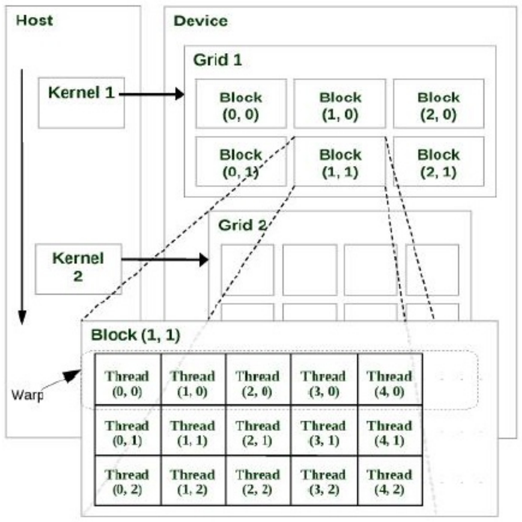


Fig. 3. NVIDIA CUDA showing execution of sequential code in the host and parallel execution of the kernel function in the device, from [3]

CUDA distinguishes CPU (host part) from the GPU (device part). Typically, initial operations and preparations are done before proceeding to the demandingly parallel computations. CUDA deals with this by performing preparations in the host and moving the prepared data to the GPU for fast parallel computation then, moving the results of the parallel computation back to the host for further interpretation of output. The entity that connects the CPU and GPU, or makes the data movement possible, is the *kernel function.* This function is called in the host but is executed in the device [13] as illustrated in Figure 3. Usually, preparations for the data are done to maximize parallelism. Operationally, the CPU controls the flow of the application program while the GPU acts as a co-processor to the host where demanding computations are held.

4 CUDA GPU Computing and ECPe systems

4.1 Generating all possible configuration vectors

The process of finding all possible configurations for all regions of an ECPe system is illustrated by Figure 4. Input files contain information regarding the ECPe system to be simulated. Some of the necessary information regarding the ECPe system are the current configuration, membrane structure, number of possible objects, number of involved rules and Transition matrix for each region k. Configuration vectors, the initial configuration and the generated configurations, are

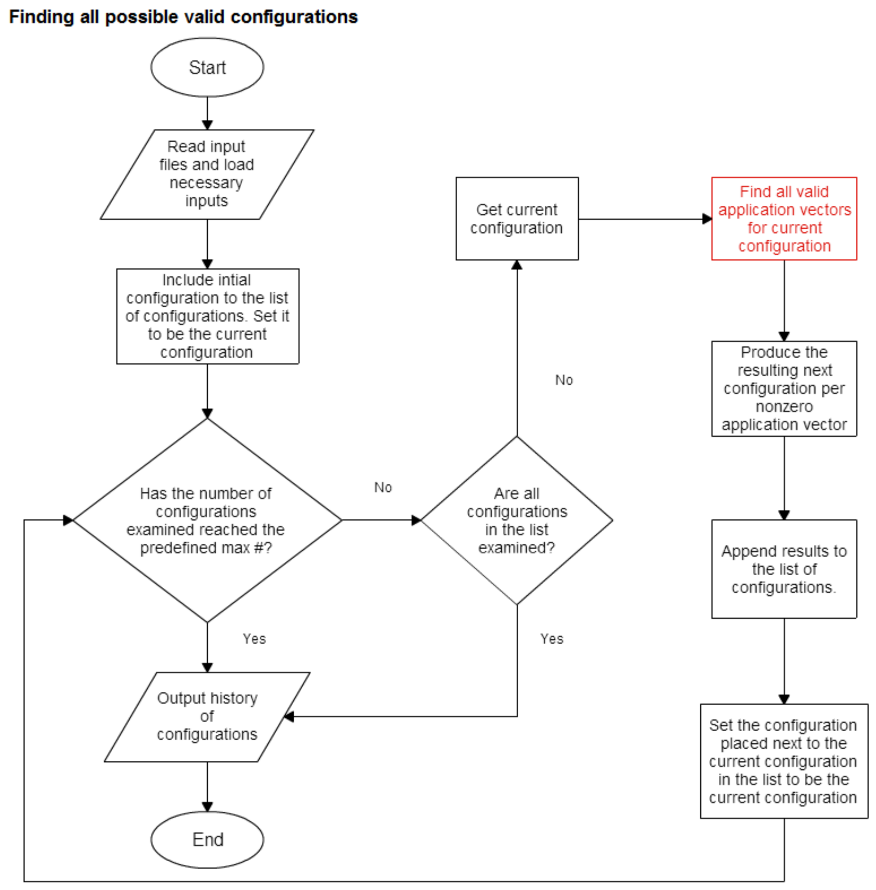

Finding all possible valid configurations
Start
Read input files and load necessary inputs
Include intial configuration to the list of configurations. Set it to be the current configuration
Has the number of configurations examined reached the predefined max #?
No
Are all configurations in the list examined?
No
Get current configuration
Find all valid application vectors for current configuration
Produce the resulting next configuration per nonzero application vector
Append results to the list of configurations.
Set the configuration placed next to the current configuration in the list to be the current configuration
Yes
Yes
Output history of configurations
End

Fig. 4. Flow chart of finding all possible configurations of an ECPe system until a halting configuration is reached or until the maximum number of iterations set by the user is reached.

written to a file. Thus, to explore each configuration, each configuration vector must be read first from a file. This vector will be labelled as the current configuration and will undergo computations, as illustrated by Figure 5 to determine all valid application vectors and to produce possible next configurations. Results are appended at the bottom of this file.

The process illustrated in Figure 4 will generate all possible next configurations until a maximum number of explored configurations have been reached, or no further configurations needs to be explored (the case of halting configuration).

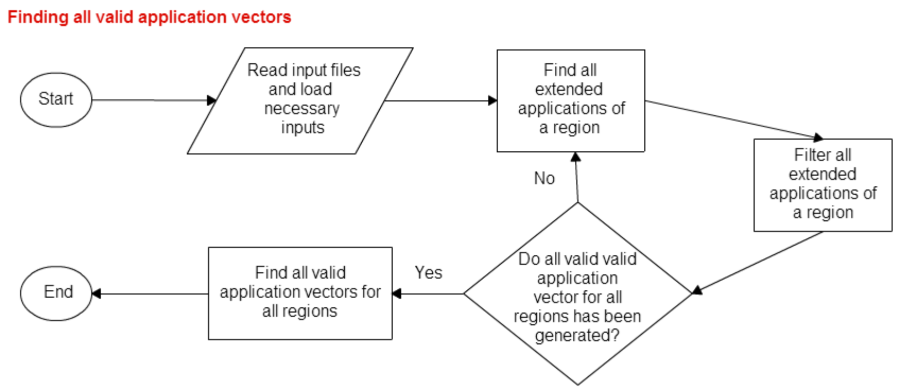

Finding all valid application vectors
Start
Read input files and load necessary inputs
Find all extended applications of a region
Filter all extended applications of a region
No
Do all valid valid application vector for all regions has been generated?
Yes
Find all valid application vectors for all regions
End

Fig. 5. Flow chart of finding all possible application vectors for all region(s) of an ECPe system. A subprocess of finding all possible configuration vectors illustrated in Figure 4, highlighted in red.

4.2 Generating all possible application vectors

The generation of all possible application vectors is guided by the methodology for forward computing as discussed in Section 2.4. It is further illustrated in Figure 5.
Steps 1-4: Preparation of input We can notice that steps 1 through 4 are done to prepare the values needed for generating application vectors. It is in Step 5 that computation starts to generate all possible application vectors. Thus, to simulate steps 1 through 4, we read from a file the necessary values needed for generating all possible application vectors.
Step 5: Finding extended application vectors through solving a system of linear equations Finding extended valid applications involves finding solutions to a system of equations. Forward computing step 5 in Section 2.4 details the system of linear equations to be solved. This can be implemented in parallel. We extend the sequential implementation of computation on ECPe systems without antiport rules and adapt ideas on parallelizing some of the processes of the implementation presented in [8]. The implementation uses localized matrix representation discussed in Section 2.4.

The following observations on the system of linear equations used in finding extended valid application vectors are helpful in solving the system in parallel. Let us take as an example the ECPe system Π illustrated in Figure 1. If we let $C_{i-1,1}$ be the initial configuration, the system of linear equations produced upon performing $\mathbf{a'}_{i,1} \cdot TM_{\Pi,1} = \mathbf{C}_{i-1,1}$ is,

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

- **Each variable's value in the resulting system of linear equations can only be a natural number** since these variables correspond to the number of applications of a rule.
- **Every equation in the system corresponds to an object condition**. Since the dimension of the Trigger matrix $TM_{\Pi,k}$ is $|IR(k) \cup R_{add,k}| \times |PO(k)|$, each equation resulting upon performing $\mathbf{a'}_{i,1} \cdot TM_{\Pi,1} = \mathbf{C}_{i-1,1}$, will give us the equation for the application of each rules (i.e. the variables in the left hand side of the equation) whose available triggering objects is in the right hand side of the equation.

$$\begin{array}{rc} a\colon & a'_{i,1}(r_{11}) + a'_{i,1}(r_{12}) + a'_{i,1}(r'_{21}) = 2 \\ \#\colon & a'_{i,1}(r'_{22}) + a'_{i,1}(Add_{11}) = 1 \\ e\colon & a'_{i,1}(r'_{21}) + a'_{i,1}(r'_{22}) + a'_{i,1}(Add_{12}) = 2 \end{array}$$

- **If k is a sending region for at least one (1) communication rule, an energy condition must be in the resulting system of equations for that region**. In our example, the resulting system of linear equations of the two regions are,
Region 1:

$$\begin{array}{rc} a\colon & a'_{i,1}(r_{11}) + a'_{i,1}(r_{12}) + a'_{i,1}(r'_{21}) = 2 \\ \#\colon & a'_{i,1}(r'_{22}) + a'_{i,1}(Add_{11}) = 1 \\ e\colon & a'_{i,1}(r'_{21}) + a'_{i,1}(r'_{22}) + a'_{i,1}(Add_{12}) = 2 \end{array}$$

Region 2:

$$\begin{array}{rc} a\colon & a'_{i,2}(r'_{21}) + a'_{i,2}(Add_{21}) = 0 \\ \#\colon & a'_{i,2}(r_{21}) + a'_{i,2}(r'_{22}) = 0 \end{array}$$

Since region 2 is a receiving region only for both symport rules, it does not have an energy condition unlike region 1 which is a sending region for both symport rules.
- **Each variable (representing application of a particular communication rule) in the energy equation occurs in exactly one other object condition**. Moreover, **only variables associated with communication rules in energy equation can occur in other object conditions**, other variables occur exactly in one equation.

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

- **Coefficients of the terms for non-energy conditions is always one** because of non-cooperative form of the evolution rules and the restriction to communication rules that only one object can be transported by a rule. **Coefficients of the terms in energy equation can be any positive integer** since communication rules must consume any amount of energy $|e| \geq 1$. The union of all rules in the non energy conditions not including the identity rules represents the set of trigger rules. For example, if (ae, in) is (ae^2, in) and $(\#e, in)$ is $(\#e^3, in)$, the resulting system of linear equations becomes,

$$\begin{gathered}
a'_{i,1}(r_{11}) + a'_{i,1}(r_{12}) + a'_{i,1}(r'_{21}) = 2 \\
a'_{i,1}(r'_{22}) + a'_{i,1}(Add_{11}) = 1 \\
2a'_{i,1}(r'_{21}) + 3a'_{i,1}(r'_{22}) + a'_{i,1}(Add_{12}) = 2
\end{gathered}$$

With the above observations, all possible application vectors can be found by solving first the energy condition. We can use each solution for solving non-energy equations since each variable representing rule application of a certain communication rule in the energy equation occurs in one and only one other object condition. For example in this equation,

$$\begin{array}{rr}
a\text{:} & a'_{i,1}(r_{11}) + a'_{i,1}(r_{12}) + a'_{i,1}(r'_{21}) = 2 \\
\#\text{:} & a'_{i,1}(r'_{22}) + a'_{i,1}(Add_{11}) = 1 \\
e\text{:} & a'_{i,1}(r'_{21}) + a'_{i,1}(r'_{22}) + a'_{i,1}(Add_{12}) = 2
\end{array}$$

The possible solutions to the energy condition are,

$a'_{i,1}(r'_{21})$	$a'_{i,1}(r'_{22})$	$a'_{i,1}(Add_{12})$
2	0	0
1	1	0
0	2	0
0	1	1
0	0	2
1	0	1

We substitute the rule application to its corresponding object to communicate. Let's take the first three solutions as an example. It will give us,

$$\boxed{\begin{gathered}
a'_{i,1}(r_{11}) + a'_{i,1}(r_{12}) + 2 = 2 \\
0 + a'_{i,1}(Add_{11}) = 1 \\
2 + 0 + 0 = 2
\end{gathered}}$$

$$\boxed{\begin{gathered}
a'_{i,1}(r_{11}) + a'_{i,1}(r_{12}) + 1 = 2 \\
1 + a'_{i,1}(Add_{11}) = 1 \\
1 + 1 + 0 = 2
\end{gathered}}$$

$$\boxed{\begin{gathered}
a'_{i,1}(r_{11}) + a'_{i,1}(r_{12}) + 0 = 2 \\
2 + a'_{i,1}(Add_{11}) = 1 \\
0 + 2 + 0 = 2
\end{gathered}}$$

Then subtract it to the number in the right hand side of the equation which corresponds to the current count of the associated communicated object.

$$\begin{array}{|c|}
\hline
a'_{i,1}(r_{11}) + a'_{i,1}(r_{12}) = 2 - 2 = 0 \\
a'_{i,1}(Add_{11}) = 1 - 0 = 1 \\
2 + 0 + 0 = 2 \\
\hline
a'_{i,1}(r_{11}) + a'_{i,1}(r_{12}) = 2 - 1 = 1 \\
a'_{i,1}(Add_{11}) = 1 - 1 = 0 \\
2 + 0 + 0 = 2 \\
\hline
a'_{i,1}(r_{11}) + a'_{i,1}(r_{12}) = 2 - 0 = 2 \\
a'_{i,1}(Add_{11}) = 1 - 2 = -1 \\
2 + 0 + 0 = 2 \\
\hline
\end{array}$$

If this yields a negative number, then this solution for energy equation is not applicable. If not, solutions for the objects constitute one extended application vector [8]. In the above illustrations the solution 0 2 0 is not a valid solution. The new linear systems of equation to solve are now,

$$\begin{array}{cc}
\begin{array}{c} a'_{i,1}(r_{11}) + a'_{i,1}(r_{12}) = 0 \\ a'_{i,1}(Add_{11}) = 1 \end{array} & \begin{array}{c} a'_{i,1}(r_{11}) + a'_{i,1}(r_{12}) = 2 \\ a'_{i,1}(Add_{11}) = 0 \end{array}
\end{array}$$

$$\begin{array}{cc}
\begin{array}{c} a'_{i,1}(r_{11}) + a'_{i,1}(r_{12}) = 1 \\ a'_{i,1}(Add_{11}) = 0 \end{array} & \begin{array}{c} a'_{i,1}(r_{11}) + a'_{i,1}(r_{12}) = 2 \\ a'_{i,1}(Add_{11}) = 2 \end{array}
\end{array}$$

$$\begin{array}{c} a'_{i,1}(r_{11}) + a'_{i,1}(r_{12}) = 1 \\ a'_{i,1}(Add_{11}) = 1 \end{array}$$

The new systems of linear equations are then solved independently of each other.

Solving Equations. As what has been observed, each equation in the system of linear equations are not always independent nor dependent of each other. In case of regions that has energy condition, each non-energy object condition are dependent on it. However, each non-energy equation iscomputation independent of each other, this is also true in regions that does not have energy condition. Thus, the approach is to solve each equation independently of each other, except for energy conditions. As discussed in the previous section, regions with energy condition solves the equation for the energy condition first. After reflecting the solutions for the energy condition to the equations for the non-energy condition, solutions to the equations for the non-energy solutions are computed.

For regions without energy condition, the computation goes straight to the independent computations for solutions to the equations for the non-energy equation. The computation for the solutions to each equation can be reduced to the Integer Partition Problem as proposed in [8].

Solving Integer Partition Problem. Integer partitioning means finding a way of writing r as a sum of positive integers which are called partitions. To use these partitions in solving a non-energy equation, the number of partitions must be less than or equal to the number of variables m in the left hand side of the equation.

combinations	sticks and pebbles	integer partitions
123	|||··	0002
124	||·|·	0011
125	||··|	0020
134	|·||·	0101
135	|·|·|	0110
145	|··||	0200
234	·|||·	1001
235	·||·|	1010
245	·|·||	1100
345	··|||	2000

Fig. 6. Interpreting $\binom{5}{3}$ as integer partitions through the sticks and pebbles analogy.

If the number of partitions is less than m, zeroes must be padded accordingly so that it represents a vector with m elements. Then permutations of the partitions must be obtained. The problem of partitioning r with m components and then obtaining permutation of the partition can be reduced to solving combinations of $\binom{n}{s}$ where $n = r + m - 1, s = m - 1$.

Combinations can be interpreted or converted into integer partitions through the sticks and pebbles analogy (*see Figure 6*). In a $\binom{n}{s}$ combinations, there are n positions, s sticks, and $n - s$ pebbles. Elements in the combinations represent the position of each sticks and the pebbles take the position where there is no stick. The number of pebbles between each stick are the elements of the integer partition.

However, solutions to each equation in the system of linear equations does not end on the generation of integer partitions. Since equations for the energy condition can contain a coefficient greater than or equal to 1, the generated integer partitions need to be further filtered to ensure that they are valid solutions to the equation for the energy condition.Given an energy equation of the form $c_1a_1+c_2a_2+...+c_ma_k = n$, we first obtain a lexicographic order of the partitions, add zeroes to these partitions accordingly, and get the permutations of the partitions. Equate these permutated partitions to the equation to get the energy solutions. Let $p_1, p_2, ..., p_m$ be the partitions of r with m components. $a_1 = \frac{c_1}{p_1}, a_2 = \frac{c_2}{p_2}, ..., a_m = \frac{c_m}{p_m}$ is an energy solution if every $a_i, i \in 1, 2, ..., m$ is a natural number.

No similar filtering is needed for the solutions to the equations for the non-energy condition since we have observed that the coefficient of the equations for the non-energy condition is always equal to 1.

Parallel Implementation To solve the equation for energy condition, the process of generation of combinations and its conversion to integer partitions (whose process is described in the previous section), are given to the threads in the GPU device.

If the number of permuted integer partitions C (also equal to the number of combinations of $\binom{n}{s}$) is less than or equal to the number of all available threads T then each thread will produce one permuted integer partition only and equate this to the energy equation to check if it is valid. If $C > T$, then each thread will produce and check $\lceil \frac{C}{T} \rceil$ partitions except for the last thread. The last thread will produce at most $\lceil \frac{C}{T} \rceil$ partitions. The number of blocks to be used is equal to $\lceil \frac{T}{M} \rceil$, where M is equal to maximum threads per block. Since the non-energy equations

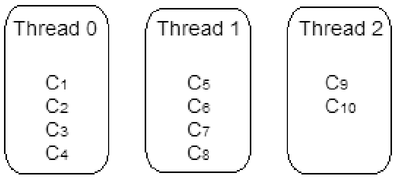


Fig. 7. When $C > T$. If there are three (3) available threads, each thread, except for the last thread, will generate four (4) integer partitions namely, $C_i, ..., C_{i+4}$and the last thread will generate three (3) integer partitions.

are independent of one another (they depend only on the energy equation) their solutions can be obtained in parallel. If there is an energy equation, the solutions to non-energy equations can be obtained in parallel after the solutions of energy equations are substituted in the system. For the implementation in GPU, a block is responsible for generating solutions of a non-energy equation. At most M threads will produce the partitions for the assigned non-energy equation to the block where they belong. If the number of non-energy equations NC is less than or equal to the number of blocks B then each block will produce solutions for one non-energy equation only. If $NC > B$, then each block will produce solutions for $\lceil \frac{NC}{B} \rceil$ number of non-energy equations except for the last block. The last block will produce solutions for at most $\lceil \frac{C}{T} \rceil$ number of non-energy equations.
Step 6: Filter extended application vectors. If the system has an energy equation, the extended application vectors must be filtered. Filtering is done in parallel. If the number of extended application vectors E is less than or equal to the number of all available threads T then each thread will filter one extended application vector only. If $E > T$, then each thread will filter $\lceil \frac{E}{T} \rceil$ extended application vectors except for the last thread. The last thread will produce at most $\lceil \frac{E}{T} \rceil$ extended application vectors. The number of blocks to be used is equal to $\lceil \frac{E}{M} \rceil$. The analogy of job allocation to threads is similar to Figure 7.
Step 7: Finding all valid application vectors In this step other necessary values are loaded from a file as well as the extended application vectors generated from the previous steps (namely, Step 5 and 6) Identity rules at each region is removed. Thereafter, the extended application vectors are merged with each other to reflect the application of symport rules from sending regions to receiving regions

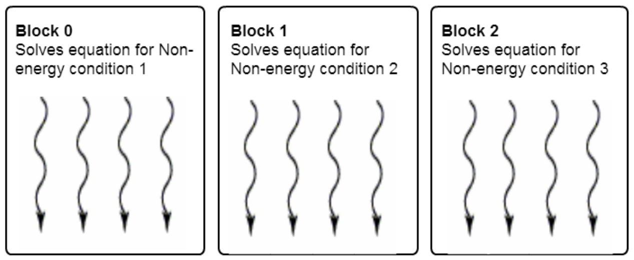

Block 0
Solves equation for Non-energy condition 1
Block 1
Solves equation for Non-energy condition 2
Block 2
Solves equation for Non-energy condition 3

Fig. 8. When $NC \leq B$. If there are three (3) available blocks and equation for non-energy conditions, each block will generate solutions to each equations utilizing the threads allocated for each block to generate integer partitions similar to

since the computation done are local to each region only. This will give us all the valid application vectors of region k at time i. This step is done in the host only, using computing and memory resources of the host only.

5 Space and Communication Requirements

STEP	**HOST ALLOCATION**	**HOST TO DEVICE**	**DEVICE ALLOCATION**	**DEVICE TO HOST**
Finding solutions for the energy equation in the system of linear equations	1 solcount ×left		(C ×left) 1 solcount ×left	1 solcount ×left
Finding solutions for non-energy equation in the system of linear equations	TM_rows ×TM_cols NEqCount × TM_rows ×2	NEqCount ×2	NEqCount × TM_rows × 2 NEqCount × validCount NEqCount NEIPsize	
Finding extended application vectors	TM_rows × extAppVecCount 1		TM_rows × extAppVecCount 1	TM_rows × extAppVecCount 1

Table 1. Memory allocations in Host and Device together with the communication between them if there is an energy equation.

Tables 1 and 2 summarizes the space requirements and size of message communicated between host and device. Note that in case of allocation, the value one

STEP	**HOST ALLOCATION**	**HOST TO DEVICE**	**DEVICE ALLOCATION**	**DEVICE TO HOST**
Finding solutions for non-energy equation in the system of linear equations	TM_rows× TM_cols	NEeqCount×2	NEeqCount× NEIPsize	
Finding extended application vectors	TM_rows			

Table 2. Memory allocations in Host and Device together with the communication between them if no energy equation.

(1) on the table implies that an extra variable is allocated to hold values for use as counter or flag. A value 1 is also used as signal for communication between host and device. In Tables 1 and 2, cells without values indicate no allocation or communication. The variables used are defined as follows:

- $membraneCount$ is the number of membranes in the system
- IR is the number of involved rules in the region
- TM_rows is the number of rows of the trigger matrix
- TM_cols is the number of columns in the trigger matrix
- $NEeqCount$ is the number of non-energy equations. For sender regions, this is equal to $TM\_rows - 1$.
- $left$ is the number of terms(partitions) in the left-hand side of the energy equation
- $right$ is the multiplicity of an object at the right-hand side of the energy equation.
- $C = \frac{(left+right-1)!}{(right)!(left-1)!)}$ is the number of integer partitions of right hand side of the energy equation (multiplicity of energy in the system) with components equal to the number of partitions in the left hand side.
- $solcount = O(C)$ is the number of valid solutions for the energy equation
- $validCount$ is the number of energy solutions for the system of linear equations.
- $extAppVecCount = O(C^{NEeqCount})$ is the number of extended application vectors of the region
- $NEIPsize$ is the summation of combinations of $\binom{i}{j}$ of each non-energy equation, where $i = m2 + r2 - 1$, $j = m2$, $m2$ being the new number of partitions in the left-hand side of the equation, and $r2$ being the new right-hand side after the energy solutions are substituted.
- $validExtAppCount = O(C^{NEeqCount})$

In the variables given, the first six items are variables that is only dependent on rules and membrane structures for a given P system, while the rest are variables that are also dependent on configuration. Consequently, this means that the rest of the variables excluding the first six potentially changes every time a new configuration is explored.

From these tables, it can be observed that the required variables for regions without energy equations are significantly less than regions with energy equations.

This is expected since regions without energy equations only act as receivers and need not be concerned with the number of application of communication rules. Thus, we only analyze the requirement for sending regions where energy equations needs to be addressed.

Focusing on sender regions, Table 1 shows that the most expensive communication step occurs upon accomplishment of finding extended valid application vectors where the set of all valid application vectors will be communicated from host to device as a matrix with size at most $C^{NEeqCount} \times TM\_cols$.

The required memory space in host in finding all valid extended application vectors (i.e. Steps 1-6 of the methodology for forward computing described section 2.4) when the region has energy is,

$$E_h = C \times left + TM\_rows \times TM\_cols + (NEeqCount \times TM\_rows \times 2)$$
$$+(TM\_rows \times C^{NEeqCount} \times 2) + 3$$

The needed memory space in host when finding all valid application vectors(i.e. Step 7 of the methodology for forward computing described section 2.4) is,

$$M = membraneCount \times (membraneCount + IR + IR \times validAppVecCount)$$

From this, we can conclude that the upperbound of the memory needed in host is $O((E_h + M))$.

On the other hand, the required memory space in device in finding all valid extended application vectors (i.e. Steps 1-6 of the methodology for forward computing described in Section 2.4) when the region has energy is,

$$E_d = C \times left \times 2 + (NEeqCount \times TM\_rows \times 2) + NEeqCount \times C + NEeqCount$$
$$+NEIPsize + (TM\_rows \times C^{NEeqCount} \times 2) + 3$$

Note that, when finding all valid application vectors (i.e. Step 7 of the methodology for forward computing described in Section 2.4), there is no need to allocate memory in device.

6 Conclusion and Future Works

In this paper, we were able to present a hybrid implementation of computation for ECPe systems without antiport by employing GPUs. Our implementation makes use of matrix representation and operations discussed in [6]. To improve our implementation, the following are recommended for further study:

- **Optimize memory usage**
  Some processes in the parallel simulation makes use of arrays that may become large (depending on the number of rules and objects in a region, number of regions, and etc). An example is the padding of energy and non-energy solutions

with 0s so that all generated solutions will have the same dimension and is faithful to the size of extended application vector determined in Step 4 of the methodology for forward computing described in Section 2.4. It is primarily done to make merging easier when it comes to finding valid application vectors. However, doing so might require a significant amount of memory.

- **Accept input from P Lingua**
  It is a good characteristic of the program if it is P Lingua compatible. That is, from the definition of an ECPe system, say Π. Its corresponding P Lingua format is accepted as an input of the program. In this way, running any ECPe system in the program would be easier. The number of input files used by the program will also lessen.
- **Concurrent processes for solving energy solutions of a region and non-energy solutions of regions without energy equation**
  Our approach in solving system of linear equations is to solve first for the energy solutions if there is an energy equation in the said system of linear equations. Thus, in this way we cannot generate solutions for the non-energy equations yet. However, it can be done that while energy solutions are being generated, non-energy solutions for other regions are also being generated.
- **Extend parallel simulation of ECPe system to include antiport rules**
  Since the current implementation is done without antiport rules and there is still no existing parallel simulation of ECPe systems which include antiport rules as of the writing of this paper, it is attractive to extend the work to include antiport rules.

Acknowledgements

F.G.C. Cabarle and R.A. B. Juayong are supported by the Engineering Research and Development (ERDT) Scholarship Program. H.N. Adorna is funded by a DOST-ERDT research grant and the Semirara Mining Corporation professorial chair of the UP Diliman, College of Engineering. M.A. Martínez-del-Amor is supported by "Proyecto de Excelencia con Investigador de Reconocida Valía P08-TIC-04200" from Junta de Andalucía, and project TIN2012-37434 from "Ministerio de Ciencia e Innovación" of Spain, both co-financed by FEDER funds.

References

1. H. Adorna, Gh. Păun, M.J. 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. S. G. Akl: Adaptive and optimal parallel algorithms for enumerating permutations and combinations, *The Computer Journal,30, 5 (1987), 433436*
3. F. Cabarle, H. Adorna, M. A. Martínez-del-Amor.: Simulating Spiking Neural P Systems Without Delays Using GPUs, *International Journal of Natural Computing Research (IJNCR), Vol. 2 No. 2 pp. 19-31, 2011*

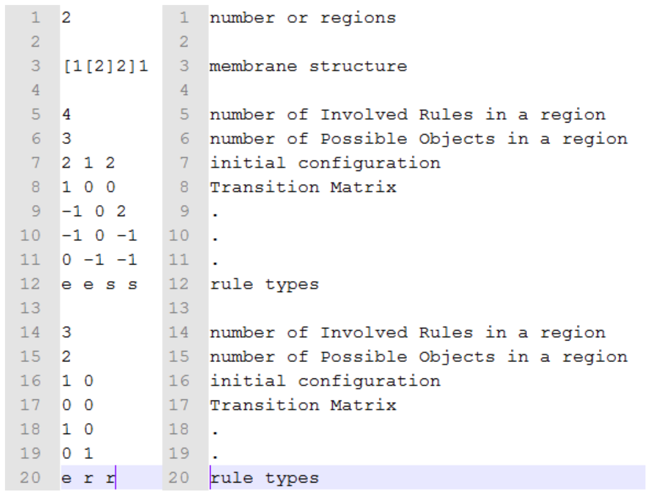
4. F. Cabarle, H. Adorna, M.A. Martínez-del-Amor, M.J. Pérez-Jiménez: Improving GPU Simulations of Spiking Neural P Systems, *Romanian Journal of Information Science and Technology Volume 15, Number 1, pp. 520, 2012.*
5. G. Ciobanu, G. Wenyuan: P Systems Running on a Cluster of Computers, *In Workshop on Membrane Computing pp. 123-139, 2003.*
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. 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*
8. R.A. Juayong, F.G.C. Cabarle, H. Adorna, M. Martínez-del-Amor: On the Simulations of Evolution-Communication P Systems with energy without Antiport Rules for GPUs, *Technical report of the 10th Brainstorming Week in Membrane Computing, Seville, Spain, Feb 2012.*
9. G. D. Knott: A Numbering System for Combinations. *Comm. ACM, Vol. 17, No. 1, pp. 45-46, January 1974.*
10. C. Mifsud, Algorithm 154: combination in lexicographical order,*Communications of the ACM, p. 103. Volume 6 Issue 3, March 1963.*
11. J.D. Owens, M. Houston, D. Luebke, S. Green, J.E. Stone, J.C. Phillips: GPU Computing, *Proceedings of the IEEE, Vol. 96 No. 5 pp. 879-899, 2008*
12. 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)
13. Gh. Păun, M.J. Pérez-Jiménez: Spiking Neural P Systems. Recent Results, Research Topics. Algorithmic Bioprocesses, Natural Computing Series. Springer. pp. 273-291. (2009)
14. Gh. Păun, M.J. Pérez-Jiménez: Solving Problems in Distributed Way in Membrane Computing: dP System, *Int. J. of Computers, Communication and Control, Vol. 5 No. 2 pp. 238-250, 2010*
15. A.E. Porreca, A. Leporati, G. Mauri, C. Zandron: Introducing a space complexity measure for P systems, *Intern. J. Computers, Communications and Control, Vol. 4 No. 3 pp. 301310, 2009.*

Appendix

Assumptions on input files

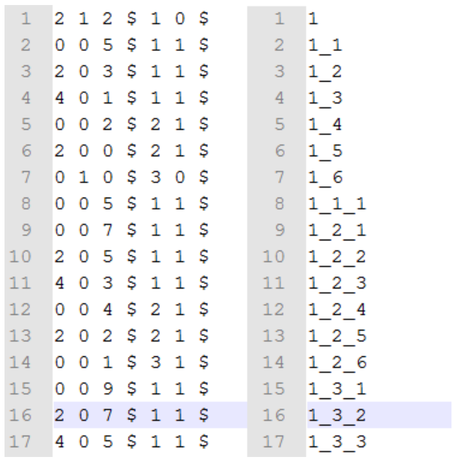
Figure 9 and 11 shows the file format for all the necessary input files. Given below are the assumptions for the input files needed in our implementation.

- A total order for objects is assumed for all of the vectors and matrices used. For example, if the correspondence of possible objects in the Transition matrix is $< a, \#, e >$, then the correspondence of possible objects in the Trigger matrix must also be $< a, \#, e >$.
- A total order for rules is assumed for all of the vectors and matrices used. For example, if the correspondence of involved rules in the Transition matrix is $< r_{11}, r_{12}, r'_{21} >$, then the correspondence of involved rules in the Trigger matrix must also be $< r_{11}, r_{12}, r'_{21} >$.
- If a region involves symport rule(s), the correspondence of the rule(s) is assumed to be at the latter part of the vectors and matrices defined for the region. For example, if r_{11}, r_{12}, r'_{21}, r'_{22} are the rules involved in region k, and r'_{21}, r'_{22} are symport rules, The total order should be $< r_{11}, r_{12}, r'_{21}, r'_{22} >$ or $< r_{11}, r_{12}, r'_{22}, r'_{21} >$.
- If a region contains symport rule(s) wherein it is a receiving region, the correspondence of the said symport rules(s) is assumed to be at the last part of the vector and matrices defined for the region. For example, if r_{11}, r_{12}, r'_{21}, r'_{22}, r'_{23} are the involved rules in region k, and r'_{21} is the symport rules where in region k is a sending region and r'_{22}, r'_{23} are symport rules where in region k is a receiving region. The total order should be, $< r_{11}, r_{12}, r'_{21}, r'_{22}, r'_{23} >$ or $< r_{11}, r_{12}, r'_{21}, r'_{23}, r'_{22} >$.



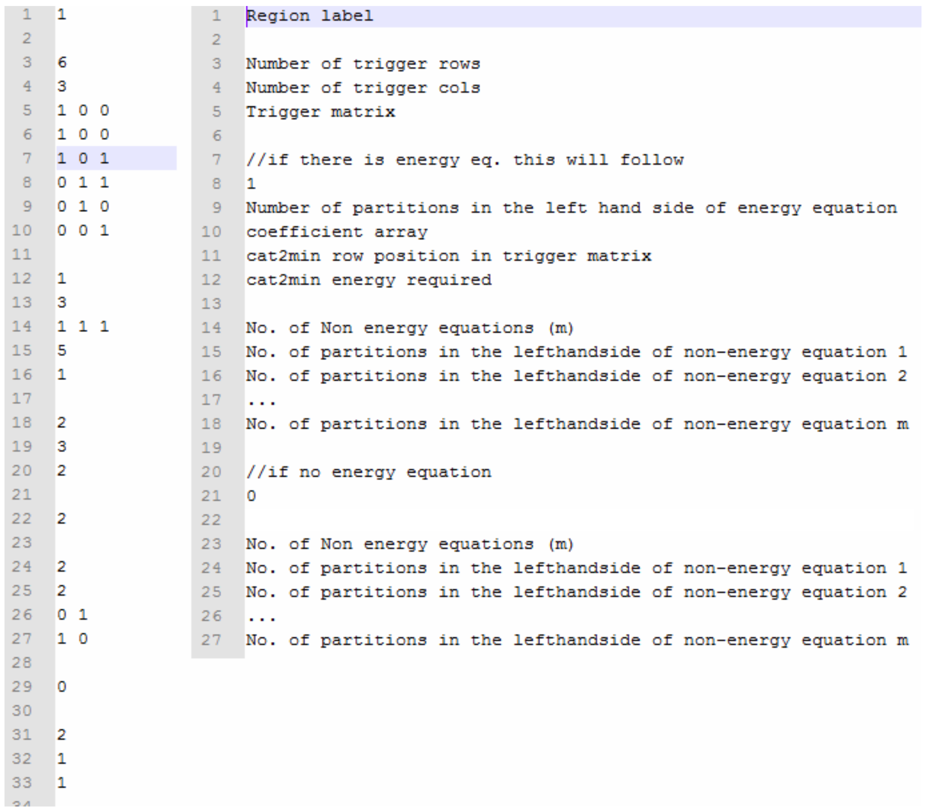
```
2             number or regions

[1[2]2]1      membrane structure

4             number of Involved Rules in a region
3             number of Possible Objects in a region
2 1 2         initial configuration
1 0 0         Transition Matrix
-1 0 2        .
-1 0 -1       .
0 -1 -1       .
e e s s       rule types

3             number of Involved Rules in a region
2             number of Possible Objects in a region
1 0           initial configuration
0 0           Transition Matrix
1 0           .
0 1           .
e r r         rule types
```

Fig. 9. Input file for generating all possible next configuration (*trans_file.txt*)



```
2 1 2 $ 1 0 $      1
0 0 5 $ 1 1 $      1_1
2 0 3 $ 1 1 $      1_2
4 0 1 $ 1 1 $      1_3
0 0 2 $ 2 1 $      1_4
2 0 0 $ 2 1 $      1_5
0 1 0 $ 3 0 $      1_6
0 0 5 $ 1 1 $      1_1_1
0 0 7 $ 1 1 $      1_2_1
2 0 5 $ 1 1 $      1_2_2
4 0 3 $ 1 1 $      1_2_3
0 0 4 $ 2 1 $      1_2_4
2 0 2 $ 2 1 $      1_2_5
0 0 1 $ 3 1 $      1_2_6
0 0 9 $ 1 1 $      1_3_1
2 0 7 $ 1 1 $      1_3_2
4 0 5 $ 1 1 $      1_3_3
```

Fig. 10. Output file of all possible next configurations Left: list of all possible next configurations(conf.txt), Right: label of all possible next configurations(*conf_index.txt*)



```
1

6
3
1 0 0
1 0 0
1 0 1
0 1 1
0 1 0
0 0 1

1
3
1 1 1
5
1

2
3
2

2

2
2
0 1
1 0

0

2
1
1
```

```
Region label

Number of trigger rows
Number of trigger cols
Trigger matrix

//if there is energy eq. this will follow
1
Number of partitions in the left hand side of energy equation
coefficient array
cat2min row position in trigger matrix
cat2min energy required

No. of Non energy equations (m)
No. of partitions in the lefthandside of non-energy equation 1
No. of partitions in the lefthandside of non-energy equation 2
...
No. of partitions in the lefthandside of non-energy equation m

//if no energy equation
0

No. of Non energy equations (m)
No. of partitions in the lefthandside of non-energy equation 1
No. of partitions in the lefthandside of non-energy equation 2
...
No. of partitions in the lefthandside of non-energy equation m
```

Fig. 11. Input file for generating extended application vectors (*find_extended_app.txt*)

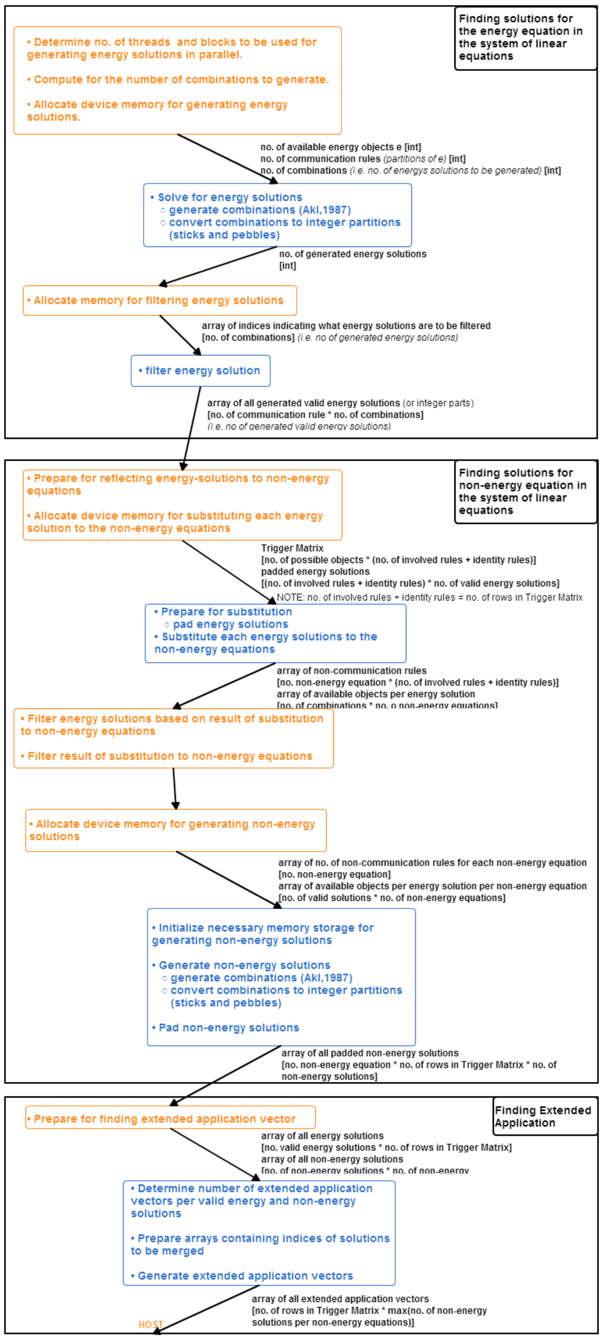

Finding solutions for the energy equation in the system of linear equations
• Determine no. of threads and blocks to be used for generating energy solutions in parallel.
• Compute for the number of combinations to generate.
• Allocate device memory for generating energy solutions.
no. of available energy objects e [int]
no. of communication rules (partitions of e) [int]
no. of combinations (i.e. no. of energys solutions to be generated) [int]
• Solve for energy solutions
○ generate combinations (Akl,1987)
○ convert combinations to integer partitions (sticks and pebbles)
no. of generated energy solutions [int]
• Allocate memory for filtering energy solutions
array of indices indicating what energy solutions are to be filtered
[no. of combinations] (i.e. no of generated energy solutions)
• filter energy solution
array of all generated valid energy solutions (or integer parts)
[no. of communication rule * no. of combinations]
(i.e. no of generated valid energy solutions)
Finding solutions for non-energy equation in the system of linear equations
• Prepare for reflecting energy-solutions to non-energy equations
• Allocate device memory for substituting each energy solution to the non-energy equations
Trigger Matrix
[no. of possible objects * (no. of involved rules + identity rules)]
padded energy solutions
[(no. of involved rules + identity rules) * no. of valid energy solutions]
NOTE: no. of involved rules + identity rules = no. of rows in Trigger Matrix
• Prepare for substitution
○ pad energy solutions
• Substitute each energy solutions to the non-energy equations
array of non-communication rules
[no. non-energy equation * (no. of involved rules + identity rules)]
array of available objects per energy solution
[no. of combinations * no. o non-energy equations]
• Filter energy solutions based on result of substitution to non-energy equations
• Filter result of substitution to non-energy equations
• Allocate device memory for generating non-energy solutions
array of no. of non-communication rules for each non-energy equation
[no. non-energy equation]
array of available objects per energy solution per non-energy equation
[no. of valid solutions * no. of non-energy equations]
• Initialize necessary memory storage for generating non-energy solutions
• Generate non-energy solutions
○ generate combinations (Akl,1987)
○ convert combinations to integer partitions (sticks and pebbles)
• Pad non-energy solutions
array of all padded non-energy solutions
[no. non-energy equation * no. of rows in Trigger Matrix * no. of non-energy solutions]
Finding Extended Application
• Prepare for finding extended application vector
array of all energy solutions
[no. valid energy solutions * no. of rows in Trigger Matrix]
array of all non-energy solutions
[no. of non-energy solutions * no. of non-energy
• Determine number of extended application vectors per valid energy and non-energy solutions
• Prepare arrays containing indices of solutions to be merged
• Generate extended application vectors
array of all extended application vectors
[no. of rows in Trigger Matrix * max(no. of non-energy solutions per non-energy equations)]
HOST

Fig. 12. Flow and communication between host and device for Step 5: Finding extended valid application vectors. (If region has an energy condition)

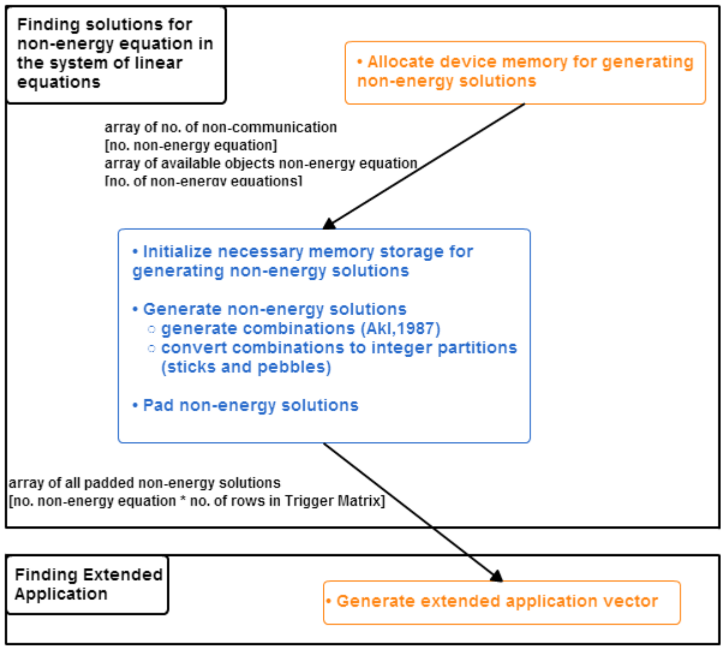

Finding solutions for non-energy equation in the system of linear equations
• Allocate device memory for generating non-energy solutions
array of no. of non-communication [no. non-energy equation]
array of available objects non-energy equation [no. of non-energy equations]
• Initialize necessary memory storage for generating non-energy solutions
• Generate non-energy solutions
○ generate combinations (Akl,1987)
○ convert combinations to integer partitions (sticks and pebbles)
• Pad non-energy solutions
array of all padded non-energy solutions [no. non-energy equation * no. of rows in Trigger Matrix]
Finding Extended Application
• Generate extended application vector

Fig. 13. Flow and communication between host and device for Step 5: Finding extended valid application vectors. (If region does not have an energy condition)

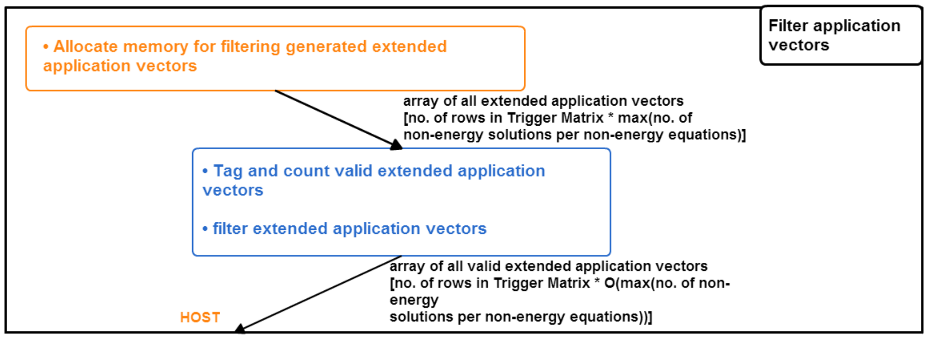

Filter application vectors
• Allocate memory for filtering generated extended application vectors
array of all extended application vectors [no. of rows in Trigger Matrix * max(no. of non-energy solutions per non-energy equations)]
• Tag and count valid extended application vectors
• filter extended application vectors
array of all valid extended application vectors [no. of rows in Trigger Matrix * O(max(no. of non-energy solutions per non-energy equations))]
HOST

Fig. 14. Flow and communication between host and device for Step 6: Filtering extended valid application vectors.