Using P System with GPU Model to Design and Implement a Public Key Cryptography

Salah Zaher
Faculty of computer & information
Cairo University
Cairo, Egypt

Amr Badr
Faculty of computer &information
Cairo University
Cairo, Egypt

Ibrahim Farag
Faculty of computer & information
Cairo University
Cairo, Egypt

Tarek AbdElmaged
Egyptian Governmental Security Consultant
Cairo, Egypt

ABSTRACT
Simulators are limited by the available resources on the GPU as well as the CPU. Simulation of P systems with active membrane using GPUs is a new concept in the development of applications for membrane computing. P systems are an alternative approach to extract all performance available on GPUs due to its parallel nature. In this paper, a design and an implementation of a simulator for a cryptography system using GPU in a P system frame is presented. Then a comparative study is conducted concerning the performance of the GPU model and the CPU model in terms of the needed time to perform encryption /decryption processes. The results show that the proposed GPU system can help in enhancement of encryption /decryption algorithm running in membrane environment.

Keywords
GPU, CPU, Membrane Computing, P system.

1. INTRODUCTION
Membrane computing (or cellular computing) is an emerging branch within natural computing that was introduced by Gh. Paun [1]. The main idea is to consider biochemical processes taking place inside living cells from a computational point of view, in a way that gives us a new nondeterministic model of computation by using cellular machines. Up to now, it has not been possible to have implementations neither in vivo nor in vitro of P systems, so handling and analysis of these devices are performed by simulators. [2].

Driven by the video games market, programmable GPUs (Graphics Processing Units) have evolved into a highly parallel, multithreaded, many core processor. They were designed to accelerate graphics applications, which transform three-dimensional data (coordinates of triangle vertices) into pixels that are displayed on a screen, using for this task programming interfaces such as OpenGL and DirectX. The massively parallel nature of graphics applications and its arithmetic intensity leads the researches to explore mapping more general non-graphics applications onto the GPU, creating a new programming field called GPGPU (General-Purpose on GPUs) [3] . Parallel computation on clusters is the traditional environment to speed-up parallel applications. Particularly, many simulators of P systems have been designed for clusters of computers. However, this computation is relatively expensive and it is available for organizations that have enough resources to buy and maintain those clusters. Nowadays, there are other cheaper solutions in the computer market that provides parallel environments.

Among these solutions, the newest generations of graphics processor units (GPUs) are massively parallel processors which allow developing a wide range of parallel applications. It is recalled that other parallel computing platforms are being investigated, such as special hardware circuits [4]. GPUs can support several thousand of concurrent threads providing a massively parallel environment where parallel applications can obtain huge performance [5]. Current NVIDIA's GPUs, for example, contain up to 240 scalar processing elements per chip, they are programmed using C and CUDA [6], and they have low cost compared with a cluster of computers. Using the power that provides GPUs to simulate P systems with active membranes is a new concept in the development of applications for membrane computing. On the other hand, P systems are an alternative approach to extract all performance available on GPUs due to its parallel nature [7].

2. MEMBRANE COMPUTING
Membrane computing is a branch of natural computing, the broad area of research concerned with computation taking place in nature and with human-designed computing inspired by nature. Besides systems biology that tries to understand biological organisms as networks of interactions, and synthetic biology that seeks to engineer and build artificial biological systems, another approach to understanding nature as computation is the research on computation in living cells [8] ,[9]. Membrane computing abstracts computing models from the architecture and the functioning of living cells, as well as from the organization of cells in tissues, organs (brain included) or other higher order structures such as colonies of cells (e.g., bacteria).The initial goal was to learn from cell biology something possibly useful to computer science, and the area quickly developed in this direction. Several classes of computing models were defined in this context, inspired from biological facts or motivated from mathematical or computer science points of view. A number of applications were reported in the last few years in several areas biology, bio-medicine, linguistics, computer graphics, economics, approximate optimization, cryptography, etc. The models investigated in membrane computing area are called P systems.

The main ingredients of a P system are (i) the membrane structure, (ii) the mustiest of objects placed in the compartments of the membrane structure, and (iii) the rules for processing the objects and the membranes. Thus, membrane computing can be defined as a framework for devising cell-like or tissue-like computing models which process multiset in compartments defined by means of membranes [10].The membrane structure consisting of several membranes arranged in a hierarchal structure [11]. A

membrane structure is represented by a Venn diagram (or a rooted tree) and is identified by a string of correctly matching parentheses, with a unique external pair of parentheses corresponding to the external membrane, called the skin. A membrane without any other membrane inside is said to be elementary. The following Example from [12] illustrates the situation: the membrane structure in Fig.1 is identified by the string.

μ = [1 [2 [5] 5 [6] 6]2 [3]3 [4 [7 [8]8] 7]4] 1

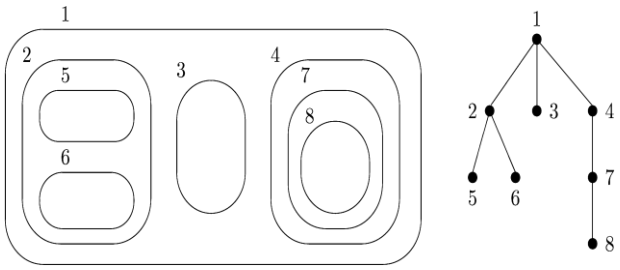

Fig 1: A membrane structure and its associated tree.

A P system with active membranes is a construct [13]

$\Pi = (V, H, \mu, w_1, w_m, R)$,

Where:

(i) M ≥ 1;
(ii) V is an alphabet;
(iii) H is a finite set of labels for membranes;
(iv) μ is a membrane structure, consisting of m membranes, labeled (not necessarily in a one-to-one manner) with elements of H; all membranes in μ are supposed to be neutral;
(v) $w_1 ... w_m$ are strings over V, describing the multisets of objects placed in the regions of μ;
(vi) R is a finite set of developmental rules;

(a) $[_h a \to v]_h^a$,

for $h \in H, \alpha \in \{+,-,0\}, a \in V, v \in V*$,

(object evolution rules, associated with membranes and depending on the label and the charge of the membranes, but not directly involving the membranes, in the sense that the membranes are neither taking part to the application of these rules nor are they modified by them);

(b) $a[_h]_h^{\alpha_1} \to [_h b]_h^{\alpha_2}$,

for $h \in H, \alpha_1, \alpha_2 \in \{+,-,0\}, a, b \in V$.

(Communication rules; an object is introduced into the membrane, maybe modified during this process; also, the polarization of the membrane can be modified, but not its label);

(c) $[_h a]_h^{\alpha_1} \to [_h]_h^{\alpha_2} b$,

for $h \in H, \alpha_1, \alpha_2 \in \{+,-,0\}, a, b \in V$

(Communication rules; an object is sent out of the membrane, maybe modified during this process; also, the polarization of the membrane can be modified, but not its label);

(d) $[_h a]_h^{\alpha} \to b$, for $h \in H, \alpha \in \{+,-,0\}, a, b \in V$

(Dissolving rules; in reaction with an object, a membrane can be dissolved, leaving its entire object in the surrounding region, while the object specified in the rule can be modified);

(e) $[_h a]_h^{\alpha_1} \to [_h b]_h^{\alpha_2} [_h c]_h^{\alpha_3}$,

for $h \in H, \alpha_1, \alpha_2, \alpha_3 \in \{+,-,0\}, a, b, c \in V$

(Division rules for elementary membranes; in reaction with an object, the membrane is divided into two membranes with the same label, may be of different polarizations; the object specified in the rule is replaced in the two new membranes by possibly new objects; all the other objects are copied into both resulting membranes);

(f) $[_{h_0} [_{h_1}]_{h_1}^+ ... [_{h_k}]_{h_k}^+ [_{h_{k+1}}]_{h_{k+1}}^-$

$... [_{h_n}]_{h_n}^-]_{h_0}^{\alpha_2} \to [_{h_0} [_{h_1}]_{h_1}^{\alpha_3}$

$... [_{h_k}]_{h_k}^{\alpha_3}]_{h_0}^{\alpha_5} [_{h_0} [_{h_{k+1}}]_{h_{k+1}}^{\alpha_4} ... [_{h_n}]_{h_n}^{\alpha_4}]_{h_0}^{\alpha_6}$,

for $n > k \geq 1, h_i \in H, 0 \leq i \leq n,$

and $\alpha_2, ..., \alpha_6 \in \{+,-,0\}$

(Division of non-elementary membranes; this is possible only if a membrane contains two immediately lower membranes of opposite polarization, + and —; the membranes of opposite polarizations are separated in the two new membranes, but their polarization can change; all membranes of opposite polarizations are always separated by applying this rule);

If the membrane labeled h_0 contains other membranes than h_1 ,..., h_n specified above, then they must have neutral charges in order to make this rule applicable; these membranes are duplicated and then become part of the content of both copies of membrane h_0; P automata , which are symport/ antiport P systems which accept strings: the sequence of objects (the terms "object" and "symbol" are used interchangeably) imported by the system from the environment during a halting computation is the string accepted by that computation (if several objects are brought in the system at the same time, then any permutation of them is considered as a substring of the accepted string; a variant, is to associate a symbol to each multiset and to build a string by such "marks" attached to the imported mustiest) [14]. A sequence of transitions constitutes a computation. A computation is successful if it halts; reaches a configuration where no rule can be applied. If an output region is specified, then the objects present in the output regions can be counted in the halting configuration and this number is the result of computation [15].

3. ARETHMATICS OPERATIONS IN MEMBRANE COMPUTING

P systems are computing models, where certain objects can evolve in parallel into a hierarchical membrane structure. Recent results show that this model is a promising framework for solving NP complete problems in polynomial time.

Let us consider a basis q ≥ 2 and

$$x = \overline{a_1 a_2 ... a_k} = a_1.q^{k-1} + a_2.q^{k-2} + ... + a_k,$$

$$a_i \in \{0,1,...,q-1\}, k.$$

be an integer in basis q.

A P-system for x (called here Arithmetical P - System - APS for short) can be defined in a natural way as,

$$\prod = (V, T, H, \mu.w_1, w_2, ..., w_{n_o}, R),$$

Where the integer n_0 is a constant fixed by the system in Computer Architecture structure n_0 = 8,16,32,64 or128.

The example of this paper uses without loss of the generality the value,

$n_0 = k+1; T = \{0,1,...,q-1\}, V \setminus T = \{f\}, H = \{1,2,...,n_0\}, \mu = [_1[_2...[n_0]n_0...]_2]_1, w_i = a_{k+1-i} (1 \le i \le k), w_i = f(k+1 \le i \le n_0.$

R is a set of rules, unspecified in this stage. Initial, all membranes have a neutral polarity.
Graphically an APS is represented in figure 2.

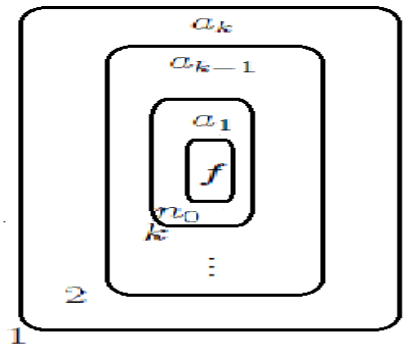

Fig 2: The structure of APS

So, in an APS each membrane contains only one object: a digit (terminal object) or f (special non terminal object); a digit from the membrane i is more significant that all digits situated in the membranes j with j < i and less significant that all digits situated in the membranes j with j > i. A f membrane is the most inner membrane or contains only f membranes. It is considered here that every APS contains at least one f membrane. Because an APS will be placed in other P systems, the outer membrane 1 of an APS will be not considered the skin.

3.1 The addition of two Arithmetical P - System (APS)

In this section it is considered that the skin contains two APS. A special object x_a will be the catalyst of operation: the addition of these APS will start in the moment when x_a is placed (somehow) in the skin as in Fig 3.

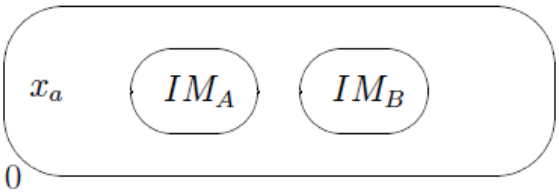

Fig 3: The addition of two APS

Also, in whole this paper the binary case (q = 2) should be considered. The generalization to an arbitrary q is easy to be accomplished.

3.2 Subtraction

Having defined the addition of two APS, the subtraction will be easy to be constructed. Let be the unsigned integers $a = \sum_{i=0}^{k-1,} a_k -_i q^i, b = \sum_{i=0}^{r-1} b_r -_i q^i.$ contained into APS A and B respectively. Supposing that a < b, then

$b - a = b - \sum_{i_0}^{k-1} a_k -_i q^i = b + \sum_{i_0}^{k-1} (q-1-a_{k-i}) q^i - \sum_{i-0}^{k-1} (q-1) q^i =$

$b + \bar{a} + 1 - q^k.$

Hence, to subtract a from b means to add b with the complement of a and with 1; finally, one unit have to be subtracted in position $k+1$.

The algorithm is:

1. $\bar{a} \to a$ (A contains the complement of a);
2. $b+1 \to b$ (The incrimination of B);
3. $a+b \to b$ (B Contains the sum $a+b+1$);
4. $b - q^k \to b$ (One unit is subtracted in the position $k+1$ of B).

Steps (1) and (2) can be accomplished in parallel; moreover, for (2) and (3) the problem is reduced to the addition of two integers. It remains to solve only steps (1) and (4).

There are necessary ($2n_0$ + 5) new objects:

$\{x_s, c', d', e', c_0, ..., c_{n_0}, x'_0, ..., x'_{n_0}\}.$

(The objects used in addition, incrimination and decrimentation are not encountered. The rules used are:

1. $[_0 x_s \to pc'c_0]_0^0.$

The first step consists in initialization of the objects which will start the four actions of the subtraction. Object p starts the incrementation of B and C starts the operation of complementarily (of A).

2. The rules used in complementarily of an APS are defined as follows:

$c'[_i]_i^0 \to [_i d']_i^0 \ (1 \le i \le n_o);$

$[_i xd' \to (1-x)c']_i^0 \quad x = 0,1, (1 \le i \le n_0);$

$[ifd' \to e'f]_i^0 \quad (1 < i \le n_0);$

$[_i e']_0^0 \to e'[_i]_i^0 \quad (1 < i \le n_0).$

Because only A has initially neutral polarity, C will penetrate the membrane 1 of A and starts the operation of complementarily.

3. When e arrives in the membrane 0, the operation of addition of these two membranes (A and B) begins:

$[_0 e' \to xa]_0^0.$

4. When the addition is performed, each dissolution of a membrane from A modifies a counter:

$$[0xci \to xci+1]_0^0 (0 \le i < n_0).$$

5. The first object f appears in the membrane 0 after dissolving of the membrane k + 1 from A (k ≥ 1).

$$[_0 c_k f \to h x'_k]_0^0.$$

(i) x'_k are new object which penetrates B until the membrane k + 1 and will start the decrementation beginning with that position. h neutralizes the other apparitions of f (if k < n):

$$[0hf \to h]_0^0.$$

(ii) Finally, the rule $[zf \to u]_0^0$ from the set (8) of addition rules will be replaced by $[_0 zhf \to u]_0^0$.

(iii) These three rules acts following priorities (iii) > (ii) > (i) because, if an object can evolve, it should evolve.

6. After the addition is finished, the action of $x'k$ begins:

$$x'_k[_i]_i^0 \to [_i x'_k]_i^0 \quad i = 1,...k+1, (1 \le k < n_0);$$

$$[_{k+1}1x'_k \to 0]_{k+1} \quad [_{k+1}0x'_k \to 1x'_{k+1}]_{k+1}^0;$$

$$[_{k+1}x'_k \to d_u]_{k+1}^0.$$

The complexity of subtraction is still constant. It depends only on the number n0 of membranes which are in an APS.

3.3 Multiplication

Time complexity: $0(n_1 \bullet n_2) = 0(n_2)$ if $n_1 = n_2 = n$. t Multiplication is implemented in a similar manner to addition, coupling a predecessor with an adder. The idea is to provide the first number to a predecessor, and perform the addition iteratively until the predecessor reaches 0. The evolution is started by the predecessor working over the first number. The predecessor activates the adder by passing a communication token. The adder is modified to use an extra backup membrane which always contains the second number.

When the adder is triggered by the predecessor, it signals the backup membrane which supplies a fresh copy of the second number to the adder, and new addition iteration is performed. At the end of the iteration, the adder sends out a token to the predecessor. The procedure is repeated until the predecessor reaches.

The P - systems which realize the product has initially the same structure with that of addition, but here the starting object is x_m .

The no terminal objects used in this operation are $\{x_m, y_0, y, z, x_0, x_1, a, b, v, v_0, v_1, p_0, p_1, 1, g, f\}$.

The rules are:

1. $[_0 x_m \to x_0 y_0]_0^0$.

2. $y_0[_1]_1^+ \to [_1 a]_1^-, [_1 a \to _2 b]_1^-,$
$b[_i]_i^0 \to [_i b]_i^- (1 < i \le n_0), [_{n_0} b \to \lambda]_{n0}^-.$

3. $[_i z]_i^- \to y, \quad y[_i]_i^- \to [_i z]_i^- (1 \le i \le n_0).$

4. $0_{[1]_1^0} \to [_1 v]_1^0, [_1 1]_1^0 \to [_1 v]_1^0 [_1]_1^+.$

5. $[_1 vj \to 0v_j]_1^0, v_j[_i]_i^0 \to [_i p_j]_i^0, [_i p_j t \to j v_t]_i^0,$
$[_i p_j f \to j]_i^0 j, t = 0,1, (2 \le i \le n_0).$

6. $[_0 x_0 f \to x_0 g]_0^0, g[_i]_i^0 \to [_i h]_i^0,$
$[_i xh]_i^0 \to g(1 \le i < n_0), [n_0 xh]_{n_0}^0 \to x_1 (x = 0,1, f),$
$[_0^0 x_0 f \to x_0]_0^0.$

7. $x_1 y[_1]_1^+ \to [_1 x_1]_1^0, [_1 x_1]_1^0 \to x_2[_1]_1^0, [x_0 x_2 \to x_a]_0^0.$

8. The addition of all APS in the membrane 0 is performed. The f result represents the product between A and B.

3.4 Division

The division of two integers is a little more complicated. Having two APS corresponding to a (nominator) and b (denominator), the algorithm of division will work in three steps:

1. At first two other APS for quotient (q) and remainder (r) will be generated;

2. By decrementing q and renew membranes 0 will be constructed, each membrane containing four APS for these integers (a, b, q, r).

3. In parallel, in each membrane 0 one verifies if the equality $a = bq + r$ holds. The membrane where this assertion is true will keep the values q and r (the APS A and B are dissolved). All the other membranes are dissolved.

The P system will have as new objects:

$\{xd, x\_, x'', z, a\_, b, a, b, a_1, b_1, q, q_1, q_2, r, r_0, r_1, r_2, r_3, l\}$.

Its initial structure is that from Figure 2, with xd instead of xa (remember, the membrane 0 is not the skin; the skin s was not drawn).

The rules are:

1. $[_0 x_d \to x' x'']_0^0, x'[_1]_1^0 \to [_1 z]_1^0, x''[_1]_1^+ \to [_1 z]_1^+.$

2. $[_1 z]_1^0 \to [_1 a]_1^+ [_1 q]_1^0, [_1 z]_1^+ \to [_1 b]_1^+ [_1 r]_1^0.$

3. The rules for decrementation of Q and R are introduced. The I value for q is a-1 and for r is b-1.

4. $[_1 q_1 x]^0_1 \to [_1 q_1 x]^+_1 [^-_1 q_2 x]^-_1 \; x \in \{0,1\},$

$[_0 [_1 q_1]^+_1 [_1 q_2]^-_1]^0_0 \to [_0 [_1 q]^0_1 [_0 [_1 q_2]^+_1]^0_0.$

5. $[_1 q_2 \to r_0 q]^+_1, [_1 r_0]^+_1 \to r_0 [_1]^+_1, r_0 [_1]^0_1 \to [_1 r_1]^0_1,$

$[r_1 \; r_1 \to r_2]^0_1.$

6. $[_1 r_2 x]^0_1 \to [_1 r_2 x]^+_1 [_1 r_3 x]^-_1, \; x \in \{0,1\},$

$[_0 [_1 r_2]^+_1 [_1 r_3]^-_1]^0_0 \to [_0 [_1 r_1]^+_1]^0_0 [_0 [_1 r_1 r]^0_1]^0_0.$

7. $[_1 f r_2]^0_1 \to [_1 0 r]^+_1.$

8. $[_1 r_1]^+_1 \to x_s x_m [_1]^-_1.$

9.
$$\begin{matrix} [ia]^-_i \to a' \\ [ib]^-_i \to b' \\ a_1 [i]^0_i \to [ia]^-_i \quad (1 \le i < n_0) \\ b_1 [i]^0_i \to [ib]^-_i \\ [_{n_0} af]^-_{n_0} \to \lambda \\ [_{n_0} bf]^-_{n_0} \to \lambda \end{matrix} \begin{bmatrix} a'b'ff \to a_1 b_1 \\ a'b'00 \to a_1 b_1 \\ a'b'11 \to a_1 b_1 \\ a'b'01 \to l \\ a'b'0f \to l \\ a'b'1f \to l \end{bmatrix}^0_0 .$$

10.
$$\begin{matrix} l[_1]^{\pm}_1 \to [_1 l]^0_1, [_1 l]^0_1 \to l\Delta, \\ l[_i]^0_i \to [_i l]^0_i, [_i l]^0_i \to l (1 < i \le n_o), \\ [_0 \Delta]^0_0 \to \lambda, [_0 \Delta x \to \Delta]^0_0 x = 0,1, f, \\ [_1 q_1 f \to l]^0_1. \end{matrix}$$

4. RSA ALGORITHM

RSA scheme is a block cipher in which the plain text and cipher text are integers between 0 and n- 1 for some n [16].

4.1 Description of the Algorithm

RSA algorithm is specified as in [17], [18], [19], [20].

4.1.1 Key setup

To setup the key material, user Alice performs the following steps:

1. Each user generates a public/private key pair by selecting two large primes at random p, q.
2. Compute N=p. q.
3. Compute $\phi(N) = (p-1).(q-1).$
4. Selecting at random the encryption key e Where $1 < e < \phi(N), \gcd(e, \phi(N)) = 1.$
5. Solve the following equation to find decryption Key d where $e.d = 1 \bmod \phi(N)$ and $0 \le d \le N.$
6. Publish their public encryption key $KU = \{e, N\}$
7. keep secret private decryption key $KR = \{d, q, p\}$

4.1.2 Encryption

To send a message m < N to Alice, the sender Bob creates the cipher text c as follows:

$$c \leftarrow m^e (\bmod N)$$

4.1.3 Decryption

To decrypt the cipher text c. Alice computes

$$c \leftarrow m^e (\bmod N)$$

4.2 Modular Exponentiation

RSA uses fast exponential algorithm called "repeated square and multiply" the algorithm repeats the following process: dividing the exponent into 2, performing square and performing an extra multiplication if exponent is odd.

Modular Exponentiation algorithm:

INPUT x, y, n: integers with $x > 0, y \ge 0, n > 1;$

OUTPUT $x^y (\bmod n).$

Mod_ exp (x, y, n);

1. If y = 0 return (1) ;

2. If y (mod 2) = 0 return (mod_exp (x2(mod n), y ÷ 2, n);

3. Return (x. mod_exp ((x2(mod n), y ÷ 2, n).

5. EXPREMINTAL RESULTS

The performance of calculations under membrane computing using GPU is compared with that using CPU in the same environment of the membrane model. The simulation was performed using laptop with Intel core2 due 2.66Ghz with RAM 3G using windows 7 with 32 bits GPU used is ATI 4650 rad eon mobility with memory dedicated 128M. The simulator was built using mathematical version8. Table 1 shows comparison for addition and subtraction between GPU & CPU. The calculations were made for 10N±10N The results of comparison for multiplication and division operations are shown in table 2& table 3 respectively where performance is calculated for 48*4N in case of multiplication and 48 /48-N is used in case of division where N ranges from 2 to 7. The results are graphed in Fig. 4 for addition and subtraction and in Fig. 5 & Fig. 6 for multiplication and division respectively. Results of Arithmetic operations with GPU are used in calculations of encryption/decryption time for RSA and compared with normal encryption/decryption time using CPU.

The results are recorded for various sizes of messages The results are shown in table 4 and the corresponding graph in figure 7.

Table 1: Addition and subtraction performance

N	GPU time (s)	CPU time (s)
2	1.00586*10^-13	0.015
3	1.00586*10^-13	0.016
4	1.00586*10^-13	0.016
5	1.00586*10^-13	0.109
6	0.11	0.578
7	0.904	4.633

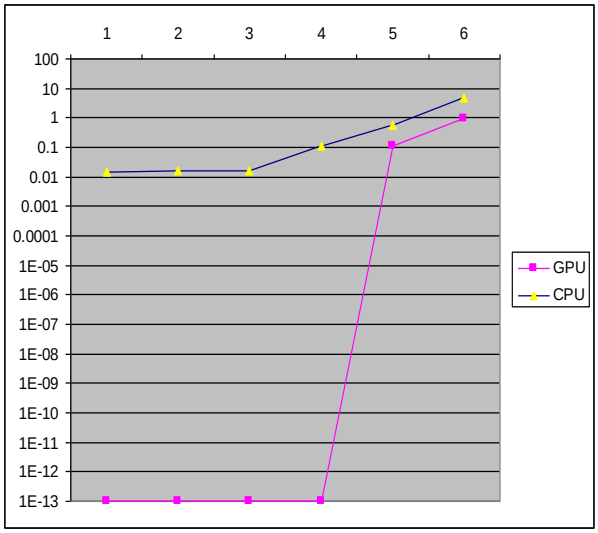

Fig 4: Addition and subtraction performance

Table 2: multiplication performance

N	GPU time (s)	CPU time (s)
2	0.031	0.157
3	0.108	0.380
4	0.271	0.930
5	1.123	2.946
6	3.602	8.920
7	17.892	27.781

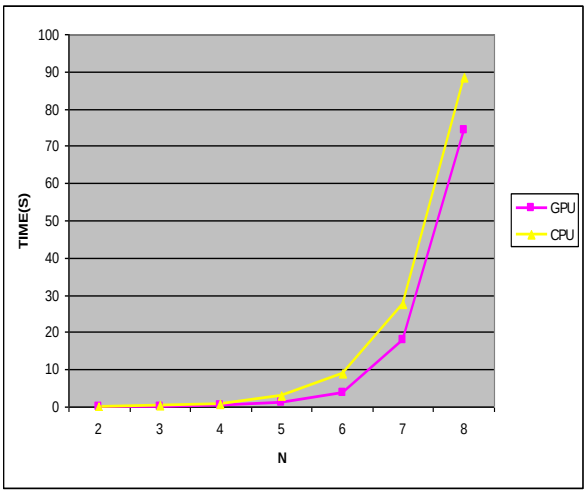

Fig 5: Multiplication performance

Table 3: division performance

N	GPU time (s)	CPU time (s)
2	0.109	0.390
3	0.281	0.920
4	1.123	2.746
5	4.602	8.720
6	18.892	27.581
7	76.425	89.404

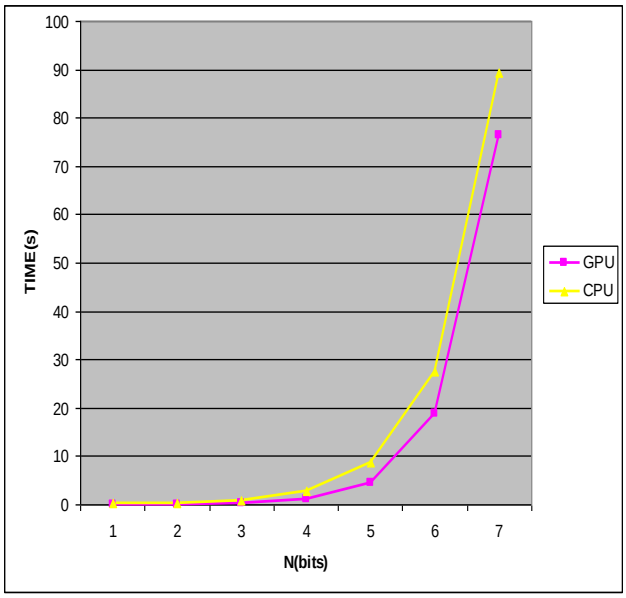

Fig 6: Division performance

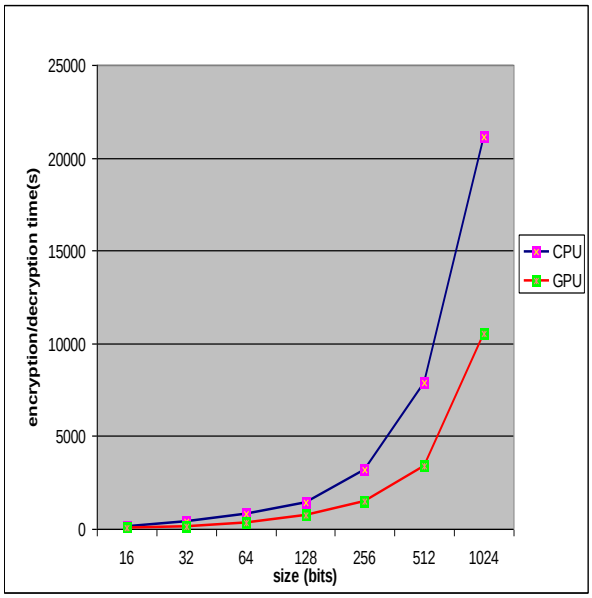

Fig 7: Encryption/Decryption time using membrane GPU versus CPU

Table 4: encryption/ decryption time of GPU membrane versus CPU

Encryptad/ decrypted massage size (bits)	Encryption/ decryption time using CPU (seconds)	Encryption/ decryption time using GPU membrane (seconds
16	140	90
32	391	128
64	798	324
128	1402	758
256	3214	1521
512	7857	3412
1024	21154	10500

6. CONCLOUSION AND FUTURE WORK

In this paper, an analysis of arithmetic operations using the membrane computing environment with both GPU and CPU is proposed. Moreover, a comparison between the performances of both techniques is conducted. The results of the comparison showed that membrane computing with GPU technique enhanced the arithmetic operations corresponding to the normal CPU technique. The cause of enhancement is using properties of parallel processing of GPU. Consequently, GPU is ideal for membrane models with massive computations. Many applications need massive calculations like encryption /decryption processes that can be enhanced using GPU with membrane model. RSA cryptography is analyzed and implemented using arithmetic calculations under GPU membranes and compared with normal CPU calculations. The results show that the RSA performance is enhanced by using GPU membrane model .

7. ACKNOWLEDGMENTS

Our thanks, respect and never ending gratitude to every one shared in this work. Great thanks to them for their effort, encouragement, advice, support and guidance.

8. REFERENCES

[1] Paun, G. 2000. Computing with membranes. Journal of Computer and System Sciences, 61, 1, 108-143, and Turku Center for Computer Science-TUCS Report No 208.

[2] Gutierrez, M.A., Naranjo, M.J. Perez, Jimenez, A. Riscos., and Nunez. 2006. Available membrane computing software. Applications of Membrane Computing, Natural Computing Series, Springer {Verlag, Chapter 15, pp. 411- 436 .

[3] NVIDIA CUDA., World Wide Web electronic publication: HHUUhttp://www.nvidia.com/UUHH cuda.

[4] Nguyen, V., Kearney, D. J., and Gioiosa. 2009. An algorithm for non-deterministic object distribution in P

systems and its implementation in hardware. Lecture Notes in Computer Science, 5391, 325-354.

[5] Hartley, T.D., Catalyurek, U., Ruiz, A., Igual, F., Mayo, R., and Ujaldon, M. 2008. Biomedical image analysis on a cooperative cluster of GPUs and multicores. ICS '08: Proceedings of the 22nd annual international conference on Supercomputing, ACM, pp. 15-25.

[6] Nickolls,J., Buck, I., Garland, M., and Skadron, K. 2008. Scalable parallel programming with CUDA. Queue, 6, 2, 40-53.

[7] Jose M. Cecilia., Gines D. Guerrero., Jose M., Garcia., Miguel A., and Martinez–del–Amor. 2009. Ignacio Perez Hurt ado, Mario J. Perez– Jimenez. A massively parallel framework using P systems and GPUs, Symposium on Application Accelerators in High Performance Computing, July.

[8] Endy, D. 2005. Foundations for engineering biology. Nature, pp 438:449–453.

[9] Dassow,G. and Paun, G. 1999. "Journal of Universal Computer Science, vol. 5, no. 2, pp 33-49".

[10] Ibarra, O.H. and Paun, G. 2007. "Membrane Computing: General View" The European Academy of Sciences.

[11] Paun, G. 2002. "Membrane Computing: An introduction" Springer Verlag, Berlin, ISBN: 3-540-42601-4.

[12] Sosik, P. and Alfonso Rodriguez-Patton. 2007. "Membrane computing and complexity theory: A characterization of PSPACE" Journal of Computer and System Sciences73, pp.137–152.

[13] Adorna, H., G. Paun, G. and PEREZ- JIMENEZ, M.J. 2010. On Communication Complexity in Evolution-Communication P Systems "Romanian Journal Of Information " Volume 13, Number 2, , pp.113-130.

[14] Paun, G. and Perez Jimenez, M.J. 2010. "Solving Problems in a distributed Way in Membrane Computing: DP systems" Int. J. of Computers, Communications & Control, ISSN 1841-9836, E-ISSN 1841-9844 Vol. V, No. 2, pp. 238-250.

[15] Paun G. 2002. "Application of Membrane Computing" Springer- Verlag, Berlin, ISBN: 3-540- 25017-4.

[16] Stallings, W. 2011. "Cryptography and Network Security Principles and Practices", fifth Edition, ISBN:13 978-0-13-705632-3.

[17] Paar. C. and jan pelzl.2010. "understanding cryptography "Spring Verlag ,Berlin , ISBN : 978 -3-642 – 04100 - 6.

[18] Esslinger, B. 2010. "The cryptool script cryptography , mathematics and more", available [http: //www.cryptool.org] .

[19] Schneier, B. 1996. "Applied cryptography Protocol , Algorithms and Code in C" ISBN:13 978-0-047-1117094, .

[20] Katz, J. and lindel, y. 2008. "introduction to modern cryptography", ISBN: 978-1-58488-551-1.