INTERNATIONAL JOURNAL OF COMPUTERS COMMUNICATIONS & CONTROL
ISSN 1841-9836, 13(3), 323-336, June 2018.

Multi-Objective Binary PSO with Kernel P System on GPU

N. Elkhani, R. C. Muniyandi, G. Zhang

Naeimeh Elkhani*, Ravie Chandren Muniyandi
Centre for Cyber Security
Faculty of Information Sciene and Technology
Universiti Kebangsaan Malaysia
43600 Bangi, Selangor, Malaysia
*Corresponding author: naeimeh.elkhani@siswa.ukm.edu.my
ravie@ukm.edu.my

Gexiang Zhang
School of Electrical Engineering
Southwest Jiaotong University
Chengdu 610031
Sichuan, P.R. China
zhgxdylan@126.com

Abstract: Computational cost is a big challenge for almost all intelligent algorithms which are run on CPU. In this regard, our proposed kernel P system multi-objective binary particle swarm optimization feature selection and classification method should perform with an efficient time that we aimed to settle via using potentials of membrane computing in parallel processing and nondeterminism. Moreover, GPUs perform better with latency-tolerant, highly parallel and independent tasks. In this study, to meet all the potentials of a membrane-inspired model particularly parallelism and to improve the time cost, feature selection method implemented on GPU. The time cost of the proposed method on CPU, GPU and Multicore indicates a significant improvement via implementing method on GPU.
Keywords: parallel membrane computing, GPU based membrane computing, kernel P system, parallel multi-objective binary PSO, parallel kernel P system-multi objective binary PSO.

1 Introduction

In the literature [1], from one point of view, methods of feature selection for classification can be divided into three families 1) methods for flat features (filter models, wrapper models, embedded models), 2) methods for structured features (graph structure) and 3) methods for streaming features. A main disadvantage of the filter approach despite its lower time consumption is the fact that it does not interact with the classifier, usually leading to worse performance results than those obtained with wrappers. However, the wrapper model comes with an expensive computational cost. An intermediate solution for researchers can be the use of embedded methods that are usually a mix of two or more feature selection methods from different origins which use the core of the classifier to establish a criterion to rank features.

From another perspective for the division of feature selection and classification methods, wide range of mixed methods are developed mainly based on evolutionary learning methods such as genetic algorithm (GA), neighborhood search like K nearest neighbor (KNN) and swarm intelligence algorithms such as particle swarm optimization (PSO). Based on our review on GA and KNN, the problems of various proposed methods can be categorized to three parts. First; e.g., pure genetic algorithm generally has limitations such as 1) slow convergence, 2) lacks of rank based fitness function and 3) being a time-consuming approach. Mixed methods of GA and

KNN were not capable to tackle with these problems completely. Second; in terms of classification accuracy, resulted accuracy in intelligent feature selection and classification algorithms varies greatly either in different datasets, also due to building unstable method overfitting risk increases dramatically when they examine on high density datasets. PSO because of first; its ability to match with graph model as genes (nodes) and define relationship between them (edge), second; higher accuracy in compared with flat (filter and wrapper) methods third; reasonable time complexity on CPU is our candidate in proposing a membrane-inspired feature selection method.Computational cost is a big challenge for almost all intelligent algorithms which are run on CPU. Recently new attempts have been started to develop parallel feature selection and classification methods such as [24] and some efforts are focused on parallelization of intelligent optimization algorithms, such as parallel genetic algorithm on CPUs/computers to identify informative genes for classification [16, 23], parallel Genetic algorithm on GPU [6, 15, 22], parallel PSO on GPU [14, 19–21, 28] and parallel processing of microarray data [13]. In this regard, our proposed membrane-inspired feature selection method should perform with an efficient time that we aimed to settle via using potentials of membrane computing in parallel processing Due to the inherent large-scale parallelism feature of membrane computing, any membrane computing inspired model can fully represent this computation model only in the case of using the parallel platform. From the beginning of introducing this model, it was a big concern in all membrane related studies. For instance, to fully implement parallelism of such membrane computing model and to support an efficient execution [25] used a platform based on reconfigurable hardware. Without parallelism, all subsequent studies face a challenge of how to make rules available in all steps of computation. In [2] a sequential computing of membrane computing, they just had an option of using one membrane and made the rules periodically available based on time-varying sequential P system. A sequential kernel P system multi objective binary particle swarm optimization feature selection and classification method proposed in the study of [8,9]. Even by using minimal parallelism of using rule; at least a rule from a set of rules in a membrane, e.g., with the active membrane; solving NP-complete problems in polynomial time through trading space for time leads to make a more efficient model of membrane computing. The architectural differences between CPUs and GPUs cause CPUs to perform better on latency-sensitive, partially sequential, single sets of tasks. In contrast, GPUs performs better with latency-tolerant, highly parallel and independent tasks. Recently, several studies attempted to utilize membrane computing to improve intelligent algorithms. For instance, multi-core processing used in the study of [17] utilized a membrane computing inspired genetic algorithm and [18] have highlighted parallelism in membrane computing in the case of solving the N-queen's problem.

As a variant of P system, kernel P system (KP system) introduced for the first time in the study of [11,12]. This variant of P system integrates most of the features of membrane computing which have been successfully used for modelling problems and are applied in various application. Generally, there is two types of the rules in KP systems: first type of rules deals with the objects to transfer them between compartments or send the objects from compartment to environment and vice versa; second type of rules deal with the membrane structure to change the topology of the compartments. Multi objective optimization refers to the problem of finding a set of values which meet the limitations and are capable of optimizing the set of values to another set of values which are the objective of optimization. PSO method itself is divided to two different approaches called single objective and multi-objective. These two approaches meet different requirements. Single objective is appropriate for the problems have only one correct solution. In versus, most of the hard problems are often confronted with multi-objective decision problem that their goal is to find the "best" solution which corresponds to the minimum or maximum value of a single objective that lumps all different objectives into one. The combination of membrane computing with optimization algorithm has been used in many studies such as [26, 27].

In this study, all the rules of KP system as rewriting and communication rule, division, input/output, link creation have used to develop the proposed KP-MObPSO model. A multi-core kernel P system multi objective binary particle swarm optimization feature selection and classification method proposed in the study of [8]. The most important attempts to parallelize membrane computing models are being done via using of graphic processing units (GPUs) [3–5, 7, 10]. All of these efforts have demonstrated that a parallel architecture is better positioned in performance than traditional CPUs to simulate P systems, due to the inherently parallel nature of them, and specifically GPUs obtain very good preliminary results simulating P systems.

2 Criteria to execute kernel P system multi objective binary PSO on GPU

The important factor in implementing a P system-based model on GPU is to attention the rate of communication between the threads, which is related to the dependencies between objects [17, 18]. Previous approaches of implementing P system on GPU did not consider this factor that exert effect on the performance of executing model on GPU. According to Figure 1, every single thread using the local memory and a thread block uses the share memory and a grid of thread blocks use global memory. The main strategy will be assigning dependent objects to the same thread for execution. Dependent objects in the proposed model are those objects that should be produced by prior rule and enter the compartment as input object to trigger the next rule. This is the reason rules are following priority for execution means those rules have higher priority should be execute first to generate the objects which are necessary to trigger the execution of other rules. Thus, in our model those objects that their existence is dependent to the existence of other objects in the compartment will execute on the same thread along with their parent objects. To design KP-MObPSO-SVM model on GPU, two important points are concerned, first, the dependency between the objects and rules to decrease the rate of communication, second; access to the lesser cost memories in the execution of threads like local and shared memory. As it is shown in the Figure 1, a single thread is used to assign objects and rules which are dependent to each other and they can use the local memory to keep the value for the objects and send the value to another rule to trigger its execution. When the execution of dependent rules is done, and completed in single threads, it will be needed to share the result of the threads and make a decision to choose the best result to continue the execution of model. To do this, a thread block which belongs to the current single threads can exchange the result via shared memory.

According to [12], A KP system of degree n is a tuple,

$$k_{\Pi} = (O, \mu, C_1, \ldots, C_n, i_0)$$

where O is a finite set of objects, called an alphabet; μ defines the membrane structure, which is a graph, (V, E), where V represents vertices indicating compartments and belongs to a set of labels $L(l_i, \ldots)$, and E represents edges; $C_i = (t_i, w_i), 1 \leq i \leq n$, is a compartment of the system consisting of a compartment type from T and an initial multiset, w_i, over O; i_0 is the output compartment, where the result is obtained (this will not be used in this study). Each rule r may have a guard g, in which case r is applicable when g is evaluated to true. Its generic form is $r\{g\}$. KP systems use a graph-like structure (similar to that of tissue P systems) and two types of rules

1. Rules to process objects: these rules used to transform object or to move objects inside compartments or between compartments. These rules are called rewriting, communication and input-output rules:

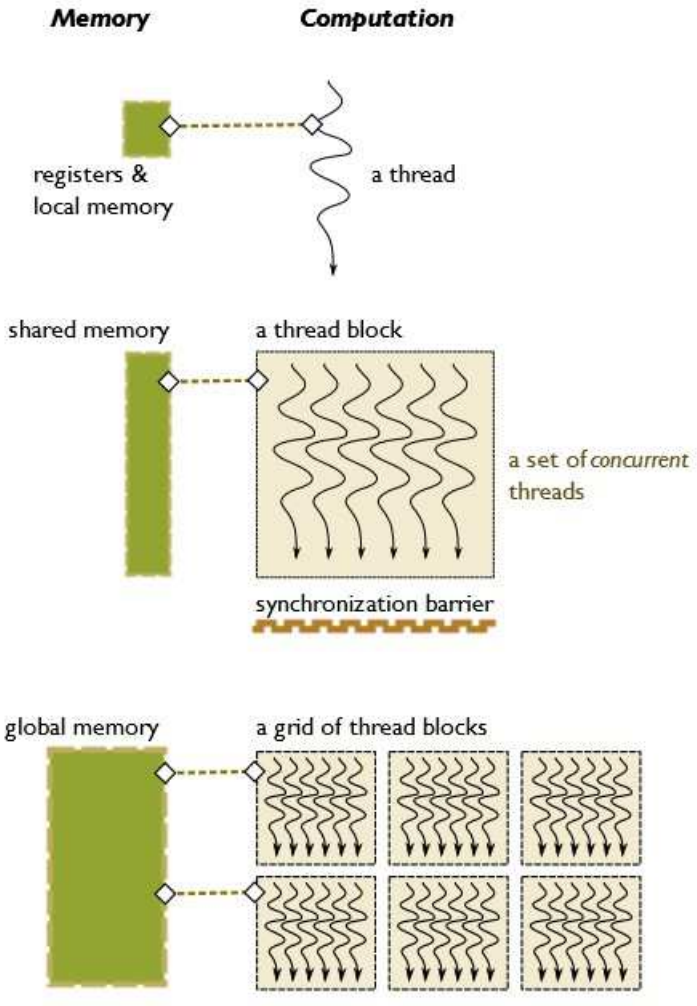

Figure 1: GPU Memory

a Rewriting and communication rule: $x \to y\{g\}$, where $x \in A^+$, $y \in A^?$, $g \in$ Finite regular Expressions FE over $(A \bigcup \bar{A})$; y at the right side defines as $y = (a_1, t_1), \ldots, (a_h, t_h)$, where $a_j \in A$ and $t_j \in L$, $1 \leq j \leq h$, a_j is an object and t_j is a target, respectively.

b The input-output rule: $(x/y)\{g\}$, where $x, y \in A^?$, $g \in$ Finite regular Expressions FE over $(A \bigcup \bar{A})$; means that x can be sent from current compartment to the environment or y can be brought from environment to the target compartment.

2. System structure rules: these rules make a fundamental change in the topology of the membranes for example with division rule on a compartment, dissolution rule on a specific compartment, make a link between compartments or dissolve the link between them. These rules are described as follow:

a Division rule: $[]_{l_i} \to_{l_i l \ldots l_i h} \{g\}$, where $g \in$ Finite regular Expressions FE over $(A \bigcup \bar{A})$; means compartment l_i can be replaced with h number of compartments. All newly created compartments inherit objects and links of l_i;

b Dissolution rule: $[]_{l_i} \to \lambda\{g\}$; means compartment l_i is not exist anymore as well as all its links with other compartments.

c Link-creation rule: $[]_{l_i}; []_{l_j} \rightarrow []_{l_i} - -[]_{l_j}\{cg\}$; means a link will be created between compartment l_i with compartment l_j. If there is more than one compartment with the label l_j, one of them will have a link with l_j non-deterministically.

d Link-destruction rule: $[]_{l_i} - -[]_{l_j} \rightarrow []_{l_i}; []_{l_j}\{cg\}$; means the existence link between l_i and l_j will eliminate and there will not be any link between them anymore. The same as link creation, if there are more than one compartment which have a link with l_i then one of them will be selected non-deterministically to apply this rule.

3 Proposed model

The entire proposed model includes two main parts, first part, features selection based on KP-MObPSO plus classification based on (kernel P system-support vector machine) KP-SVM and second part, KP-embedded feature selection/SVM classification. The first part defines modelling and implementing previous MObPSO based on KP system rules with some improvements which leads to the result consists of different set of marker genes, so called KP-MObPSO feature selection. Thereafter, an error rate calculator based on KP-SVM applied to measure the error rate of marker gene sets. To design the first part of the model, first it is needed to design improved version of KP-MObPSO on GPU. To do so, we assume there are 4 compartments each including 6 particles as Figure 2 and Figure 3. The same process will repeat for each particle as follow:

Step 1: to assign one thread for each gene of dataset (which is including of 100 genes) first we need to allocate memory in the Host for dataset of genes. Each gene keeps the values of six samples, therefore an array of threads of at least size 6 is needed. Moreover, some other threads are defined to save the values will be generated in the processing steps on the GPU and will return back to the Host. Step 2: after allocating Host memory to each gene in the dataset, genes are needed to transfer to Device memory to execute on GPU. Step 3: by assigning random number of genes inside each particle, the main process will start by executing KP-MObPSO-SVM. The kernel defined as "addkernel" will add the genes inside the particles and will execute the rules on each thread. The sets of rules called "subgraph" and "Mycost" carry the dependent rules as R1 to R10 (Table 1). Thus, these rules will execute on each single threads of genes. Local memory will keep the value of all variables which are defined as objects and all the rules have access to the same local memory to pull and push the value of objects. After that, the value of object "Fit" for each thread and the initial value for the object "pBestScore" need to save in shared memory to execute the function called "compare" according to the rules R11 and R12 to refresh the value of object "pBestScore" with the minimum value of objects "Fit". This function will implement on all threads from thread 1 to thread 4. Then, the velocity function according to the R13 to R16 will be executed on each thread with utilizing of local memory Figure **??**. Also, two sets of rules including replacing rules and decision rules will be executed on threads to initiate the cycle of particle preparation inside the compartments and restart the first part of model again till predefined iteration (it=100). Then after getting the marker genes collected by each thread, KP-SVM rules will apply to calculate the error rate for each set of marker genes. These rules are reminded in (Table 2). Step 4: at the end of first part of the model, the error rates of each set of marker genes will be calculated. According to the Table 2, (R1) is flag which is an object with default value zero. R2, flag value will rewrite to 1 if it meets the guard values including an error rate between 0 and 0.3 as well as having at least two cancerous genes indexes in marker genes. R3, q and e are the counters of normal genes and cancerous genes respectively which rewrite to a default value zero, and marker genes 2 keep a backup of marker genes resulted from first part of the model.

Table 1: KP-MObPSO Rules

R1:Rewriting $[[p, max\_c]position]_0 \rightarrow [[(position_1 \dots position_{\mathrm{n}})_1 \dots (position_1 \dots position_{\mathrm{n}})_{\mathrm{p}}]_0[[]_1[]_{\mathrm{position}}]_0$
R2:Communication $[[(position_1 \dots position_{\mathrm{n}})_1 \dots (position_1 \dots position_{\mathrm{n}})_{\mathrm{p}}]position]_0 \rightarrow$ $[(position_1 \dots position_{\mathrm{n}})_1 \dots (position_1 \dots position_{\mathrm{n}})_{\mathrm{p}}]_1]_0$
R3:Communication $[(position_1 \dots position_{\mathrm{n}}) \dots (position_1 \dots position_{\mathrm{n}})_{\mathrm{p}}]_1 \rightarrow [(position_1 \dots position_{\mathrm{n}})]p_1 \dots [(position_1 \dots position_{\mathrm{n}})_{\mathrm{p}}]]_1$ $Rulesinsideeachp : []_{\mathrm{p1}} \dots []_{\mathrm{pn}} : r4 > r5 > r6 > r7$
R4:Rewriting $[(position, a, max_c, p)] \rightarrow [NGENES], [NGENES] \rightarrow [NewNGENES, Q, c]$
R5:(Communication/Rewriting) $[(NewNGENES, c, p, a)] \rightarrow [C], [C] \rightarrow [sumdiss], [a] \rightarrow [snr]$ $[snr] \rightarrow [sumsnr], [sumdiss, sumsnr] \rightarrow [FIT]$
R6:Link creation $[]position - - - - []master$
R7:(Communication/Rewriting) $[FIT] \rightarrow [pBestScoren]master, [Q^n] \rightarrow [Q^n]master$
R8:Division $[[P_{\mathrm{n}}][pBestScore_{\mathrm{n}}, Q_{\mathrm{n}}]_{\mathrm{master}}]_1]_0 \rightarrow$ $[P]_1 \dots [P]_{\mathrm{n}}[pBestScore_{\mathrm{n}}, Q_{\mathrm{n}}, gBestScore]_{\mathrm{master}}]_1[[]_{\mathrm{p1}} \dots [P]_{\mathrm{n}}$ $[fitness, pBest, gBest, Velocity, c1, c2, w, Vmax, s]_{\mathrm{master}}]_1]_0$
R9: Membrane Dissolution $[[]P_1 \dots []P_n[]_{\mathrm{master}}]_1]\theta \rightarrow \lambda$
R10: Link Creation $[[]P_1 \dots []P_n[pBestScoren]_{\mathrm{master}}]_1]_0, [[]P_1 \dots []P_n[fitnessn]_{\mathrm{master}}]_1]_0 \rightarrow$ $[]P_1 \dots []P_n[pBestScoren]_{\mathrm{master}}]_1]_0 - - - - - [[]P_1 \dots []P_n[fitnessn]_{\mathrm{master}}]_1]_0$
R11: Communication/rewriting $[[[pBestScore_n]_{\mathrm{master}}]_1]_0 \rightarrow [[fitness_n]_{\mathrm{master}}]_1]_0, [[[pBest_n]_{\mathrm{master}}]_2]_0 \rightarrow$ $[[[fitness_n]_{\mathrm{master}}]_2]_0 < [[[pBestScore_n]_{\mathrm{master}}]_1]_0, 1 \leq n \leq p$ $[[[gBestScore]_{\mathrm{master}}]_1]_0 \rightarrow [[pBestScore_n]_{\mathrm{master}}]_1]_0, [[[gBest_n]_{\mathrm{master}}]_2]_0 \rightarrow$ $[[[pBestScore_n]_{\mathrm{master}}]_1]_0 < [[[gBestScore]_{\mathrm{master}}]_1]_0, 1 \leq n \leq p$ $[[[converge]_{\mathrm{master}}]_1]_0 \rightarrow in[[[[pBestScore]]_{\mathrm{master}}]_1]_0$
R12:Communication $[[[position]]_1]_0 \rightarrow [[[position]_{\mathrm{master}}]_1]_0$
R13: Communication/rewriting $[[[position_n, c1, c2, c, w, pBest, gBest, p, max_c, rand]_{\mathrm{master}}]_1]_0 \rightarrow [[[Velocity]_{\mathrm{master}}]_2]_0$ $[[[Velocity]_{\mathrm{master}}]_2]_0 \rightarrow [Vmax]_{\mathrm{master}}]_1, Velocity > Vmax$ $[[[Velocity]_{\mathrm{master}}]_1]_0 \rightarrow [-Vmax]_{\mathrm{master}}]_2, Velocity < -Vmax$ $[[[[position]]_{\mathrm{master}}]_1]_0 \rightarrow, rand <= 1/(1 + exp(-2 * Velocity)$ $[[[[position]]_{\mathrm{master}}]_2]_0 \rightarrow 0, rand > 1/(1 + exp(-2 * Velocity)$
R14:Output/link creation/communication & rewrite $[[[[position]]_{\mathrm{master}}]_1]_0 \rightarrow [[position]_1]_0[[]_1]_0, [[]position]_0 \rightarrow []_1]_0 - - - -$ $[[]position]_0[[[position]]_1]_0 \rightarrow [[position]_position]_0$
R15: Division rule $[[]p1 \dots []pn[]master]12[[]p1 \dots []pn[]master]121[[]p1 \dots []pn[]master]122$
r16: Membrane dissolution $[[]p1 \dots []pn[]master]12 \rightarrow \lambda$

Table 2: KP-SVM Rules

R1:Rewriting $(s) \rightarrow find(M(:) == i), table(i) \rightarrow lenght(s), 1 \leq i \leq 100$
R2:Rewriting $markergenes(i, it+1) \rightarrow i, table(i) > it, 1 \leq i \leq 100, 1 \leq it \leq n$
R3:Rewriting $y(i,1) \rightarrow +1, max\_j \rightarrow max\_j + 1, 1 \leq i \leq 50$
R4:Rewriting $y(i,1) \rightarrow -1, max\_f \rightarrow max\_f + 1, 51 \leq i \leq 100$
R5:Rewriting $Badgenes(it+1,1) \rightarrow max\_f$
R6:Rewriting $Y(j,1) \rightarrow +1, 1 \leq j \leq max\_j * 3$ $Y(j,1) \rightarrow -1, max\_j * 3 \leq j \leq max\_j * 3 + max\_f * 3$
R7:Input $wholedata(k,1) \rightarrow a(i,j), 1 \leq i \leq 100, table(i) > it, j = 1, k \rightarrow k+1$ $wholedata(k,1) \rightarrow a(i,j), 1 \leq i \leq 100, table(i) > it, j = 3or5$ $wholedata(k,2) \rightarrow a(i,j) 1 \leq i \leq 100, table(i) > it, j = 2or4$ $wholedata(k,2) \rightarrow a(i,j), k \rightarrow k+1 1 \leq i \leq 100, table(i) > it, j = 6$
R8:Rewriting $wholedata(k,1), wholedata(k,2) \rightarrow XY, Holdout = 0.10 \rightarrow Pcvpartition$ $X(P.training), Y(P.training) \rightarrow SVMStructsvmtrain$ $SVMStruct, X(P.test) \rightarrow Csvmclassify Sum(Y(P.test)C)/P.testsize \quad errRate$ $Y(P.test), C \rightarrow conMat$ $ERR(1,1) \rightarrow errRatefirst \quad 100 \quad times \quad iteration, \quad gBestScore = inf$ $ERR(it+1,1) \rightarrow errRate$ $Constant \rightarrow ERR(it+1,1) not \quad first \quad 100 \quad times \quad iteration, \quad gBestScore \neq qinf$
R9:Rewriting $particle2 \rightarrow Markergenes(:, it+1) 1 \leq it \leq n, 0 \leq ERR(it+1,1) \leq 0.3$
R10:Division $a(i,j) \rightarrow (Particle2, it = 1), (Particle2, it = 2), \ldots, (Particle2, it = n)$

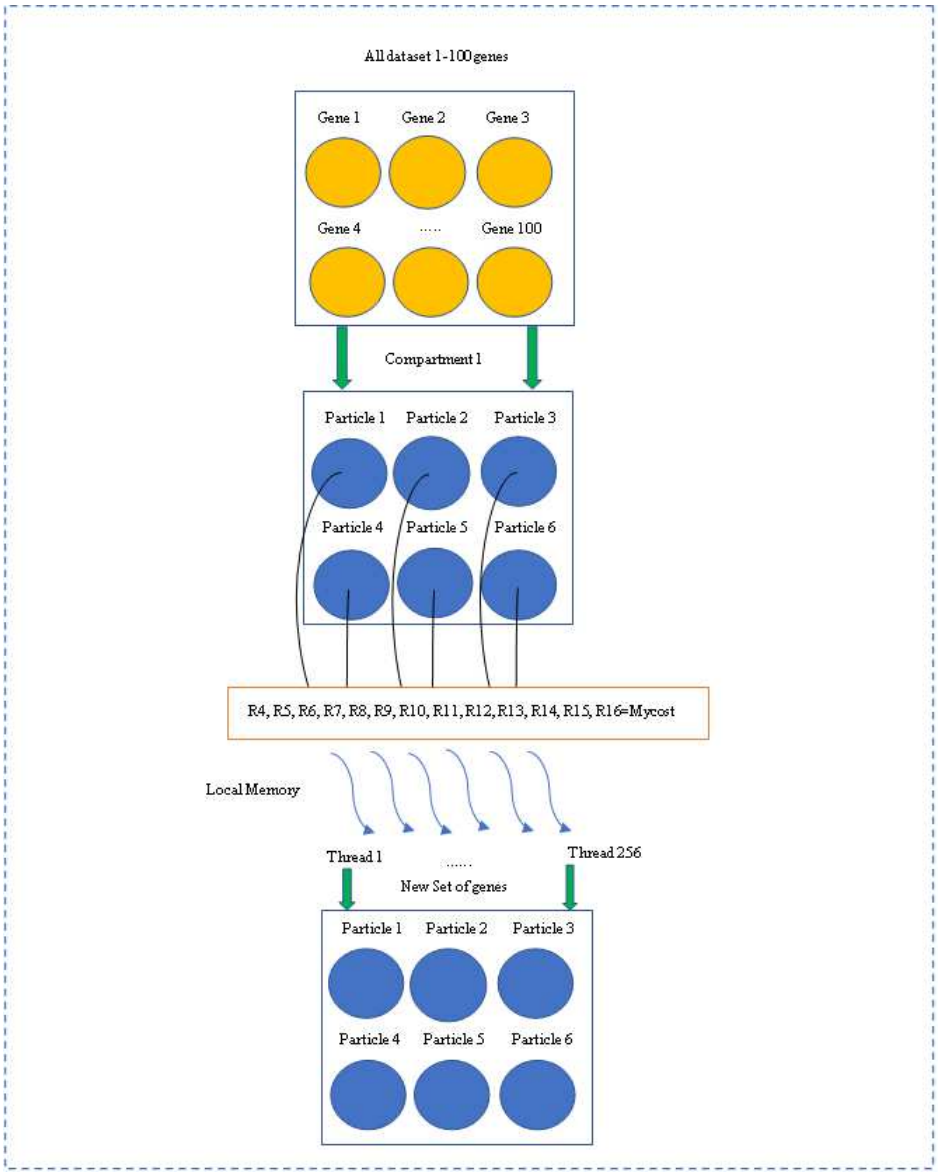

Figure 2: Designing on GPU

Step 5: elapsed time so far for feature selection executed on one particle is 5.914 Sec.

Step 6: Let assume compartment number 1 is chosen because of meeting the criteria of having better error rate. Marker genes 2 object will be used for further procedure on GPU. To implement embed feature selection method, another kernel defined as "second kernel". For each gene number from 1 to 100, rules number r4-r10 will apply to see whether entering a new gene can improve the error rate of that particle or no. R4 and R5, gives a flag to index of genes based on the type of genes whether they are belonged to normal genes or cancerous genes as +1 and -1, respectively.

In parallel, q and e which are the counters of normal genes and cancerous genes will be update. R6, the object max_j and max_f updates the number of normal and cancerous genes. R7, clear the value of q and e. R8 reserve a place for the samples of gene indexes are selected as marker genes and R9, inputs the real value of reserved samples inside a compartment called *wholedata*. *Wholedata* compartment keeps real data samples for gene indexes are already highlighted as

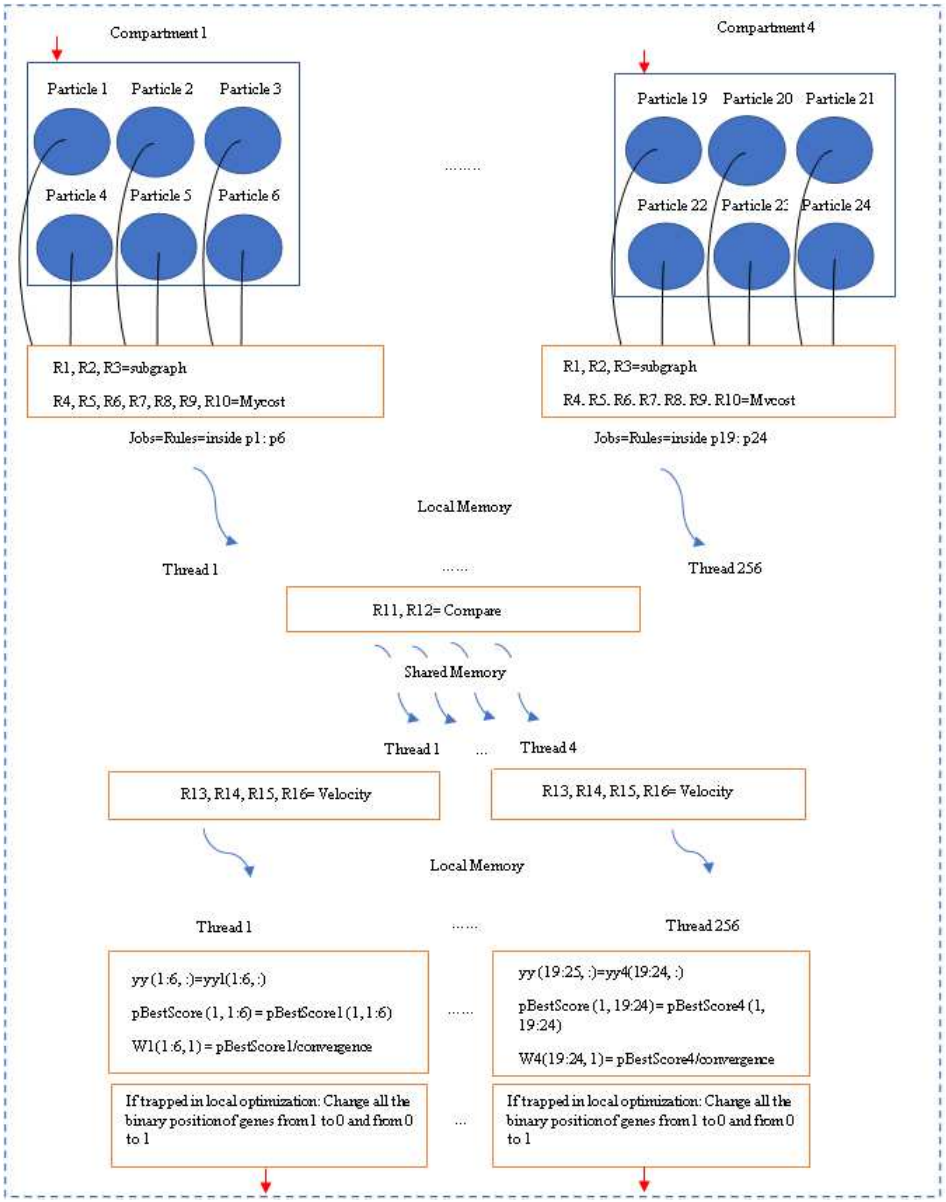

Figure 3: Continue Designing on GPU

marker genes. R10, applies an SVM package with rewriting rules to evaluate error rates. To decide whether adding a new gene can improve the error rate or no, rule number 11 compares error rates after adding each gene (from 1 to 100) with the constant error rate object resulted in the first part of modelling for each set of marker genes. If adding a new can already can improve constant error rate of that set of marker genes, the r11 will add this gene index to the set of marker genes. Otherwise if a gene value already enter to the particle cannot improve the error rate, it should be exit from the particle. Rules, R12-15 apply output and rewriting rules to eliminate gene values inside the particle. After executing the computation model for n number of iterations which each iteration represents one particle from it=1 to it=n, particles of marker genes will update by new genes and result a new set of particles as (particle 3, it=1/ particle 3, it=2/. . . / particle 3, it=n). It means by the end of embedded feature selection and classification; real data set will divide to a new compartment. Every time one gene will be added to the set of genes and error rate will calculate to decide adding the gene to the set is suitable or no. If

Table 3: Benchmark

KP-MObPSO-SVM	
Max_c =100	Maximum number of genes
P=25	Number of particles
$Max_iteration$=100	Maximum number of generating
fitness, gbestscore, pbestscore	Objects to save the results
Hardware CPU	Core(TM)i5-6200U CPU @2.30 GHz (RAM): 12.0 GB
Hardware GPU	NVIDA Geforce 680

adding the gene decrease the error rate means the gene will be kept and if increase the error rate means the gene will be removed of the chosen gene set. New value for "SUM", "mean" value, "SNR" (signal to noise) value, "dissimilarity" value, "fit" (fitness) value, "velocity" value, "std" (standard deviation) value, "gBestvalue" value. Total elapsed time for all processes of 2 particles is 13.12 Sec.

4 Evaluation and result

The objective of this study was to improve the time efficiency in implementing a membrane-based optimization algorithm. In this regard, two models as KP-MObPSO and KP-MObPSO-SVM are implemented on CPU, multicore and GPU to compare their time complexity. The comparison of the time cost between two models indicates GPU based executions can increase time efficiency of the models drastically. The benchmark of evaluation in terms of the maximum number of the genes, number of particles, maximum number of iterations, and hardware specification of CPU and GPU are explained in the Table 3. According ot Table 4 and Table 5 with the same example of 25 particles executed on multi-core and GPU with 4 workers and independent iteration respectively, time cost dropped significantly from 5.5 min to 73.87 sec in KP-MObPSO feature selection and from 15 min to 164 sec in KP-MObPSO feature selection and classification. While execution on CPU was taken 3.68 min in KP-MObPSO feature selection and 8 min in Embed KP-MObPSO-SVM feature selection and classification. Comparing CPU and multicore indicates, due to the frequent interaction between clients and workers, it takes longer time and does not lead to improvement in timely execution of multicore KP-MObPSO. Therefore, as it is shown by example, a GPU based KP-MObPSO and KP-MObPSO-SVM are more time efficient than CPU based models. Regardless of taking example, Table 3 indicates the time complexity of the parallel KP-MObPSO-SVM on GPU according to the big O calculated as $O(NM) + O(N) + O(N/P * M) + O(N/P) + O(M/2P)) + O(M/P) + O(M/(P * Q)) + O(M2/P^2) + O(M^2/2P^2) + O(Q^2/P^2) + O(2Q/P) + O(M * Q^2/P^2)$ where (N=Max number of particles, M=Max number of genes, I=Max number of iteration, Q: Max number of samples, P=Max number of processors). The time complexity of sequential KP-MObPSO-SVM is $O(NM) + O(INM)) + O(IN) + O(M/2) + O(M^2/2) + O(M/2) + O(MQ) + O(M) + O(M^2) + O(Q^2) + O(Q) + O(MQ^2) + O(N)$ where (N=Max number of particles, M=Max number of genes, I=Max number of iteration, Q: Max number of samples). $O(M^2) + O(MQ^2) and O(M^2/P^2) + O(Q^2/P^2)$ are the highest time cost for sequential and parallel KP-MObPSO-SVM respectively. It is shown how the number of processors decrease the time cost of implementing KP-MObPSO-SVM on GPU and leads to more time efficient method.

Table 4: KP-MObPSO

KP-MObPSO-SVM	
CPU	25 particles: 3.68 min (10 times of 100 iteration)
Multi-Core	25 particles: 5.5 min (4 workers)
GPU	2 particles: 5.91 Sec 25 particles: 73.87 Sec (independent iteration)

Table 5: KP-MObPSO-SVM

KP-MObPSO-SVM	
CPU	25 particles: 8 min (10 times of 100 iteration)
Multi-Core	25 particles: 15 min (4 workers)
GPU	2 particles: 13.12 Sec 25 particles: 164 Sec (independent iteration)

5 Conclusion

To design KP-MOBPSO-SVM model on GPU, two important points are concerned, first, the dependency between the objects and rules to decrease the rate of communication, second; access to the lesser cost memories in the execution of threads like local and shared memory. Thus, according to the first criteria, those objects that their existence is dependent to the existence of other objects in the compartment will execute on the same thread along with their parent objects. Based on the second criteria, objects and rules which are dependent to each other will use the local memory to keep the value for the objects and will send the value to another rule to trigger its execution. When the execution of dependent rules is done, and completed in single threads, it will be needed to share the result of the threads and make a decision to choose the best result to continue the execution of model. To do this, a thread block which belongs to the current single threads can exchange the result via shared memory. The time cost of KP-MObPSO and Embedded KP-MObPSO-SVM on GPU, CPU and Multicore are compared in the Table 3 which indicates a significant improvement in time cost via executing both KP-MObPSO and Embedded KP-MObPSO-SVM on GPU. In 25 particles, KP-MObPSO takes 3.68 min to complete a set of 100 iteration. While the multicore due to frequent visiting of client to exchange the result was not capable of improving time complexity, GPU does better. In GPU-based execution of KP-MObPSO, 25 particles take 73.87 sec to complete. Therefore, a four times improvement has happened in time efficiency. In terms of KP-MObPSO-SVM, 8 min execution time have not improved by multicore while GPU has improved the efficiency to 5-fold. The big O calculation indicates the time efficiency of the proposed KP-MObPSO-SVM improved from O(M2)+O(MQ2) in sequential method to O(M2/P2)+O(Q2/P2) in parallel execution.

Acknowledgment

The work of N. Elkhani and R. C. Muniyandi has been supported by FRGS/1/2015/ICT04/UKM /02/3, National University of Malaysia, Ministry of Higher Education, Malaysia. The work of G. Zhang was supported by National Natural Science Foundation of China (61672437, 61702428) and by Sichuan Science and Technology Program (18ZDYF2877, 18ZDYF1985, 2017GZ0159).

Bibliography

[1] Alelyani, S.; Tang, J.; Liu, H. (2013); Feature Selection for Clustering: A Review, *Data Clustering: Algorithms and Applications*, 29, 110-121, 2013.

[2] Alhazov, A.; Freund, R.; Heikenwalder, H.; Oswald, M; Rogozhin, Y.; Verlan, S. (2012); Sequential P systems with regular control, Paper presented at the *International Conference on Membrane Computing*, 2012.

[3] Cabarle, F. G. C.; Adorna, H.; Martinez-Del-Amor, M. A.; Perez-Jimenez, M. J. (2012); Improving GPU simulations of spiking neural P systems, *Romanian Journal of Information Science and Technology*, 15(1), 5-20, 2012.

[4] Cecilia, J. M.; Garcia, J. M.; Guerrero, G. D.; Martinez-del-Amor, M. A.; Perez-Hurtado, I.; Perez-Jimenez, M. J. (2009), Simulation of P systems with active membranes on CUDA, *Briefings in bioinformatics*, 11(3), 313-322, 2009.

[5] Cecilia, J. M.; Garcia, J. M.; Guerrero, G. D.; Martinez-del-Amor, M. A.; Perez-Hurtado, I.; Perez-Jimenez, M. J. (2010); Simulating a P system based efficient solution to SAT by using GPUs, *The Journal of Logic and Algebraic Programming*, 79(6), 317-325, 2010.

[6] Cano, A.; Zafra, A.; Ventura, S. (2010); Solving classification problems using genetic programming algorithms on GPUs, *Hybrid Artificial Intelligence Systems*, 17-26, 2010.

[7] Dematte, L.; Prandi, D. (2010); GPU computing for systems biology, *Briefings in bioinformatics*, 11(3), 323-333, 2010.

[8] Elkhani, N.; Chandren Muniyandi, R. (2017); A Multiple Core Execution for Multiobjective Binary Particle Swarm Optimization Feature Selection Method with the Kernel P System Framework, *Journal of Optimization*, 2017.

[9] Elkhani, N.; Muniyandi, R. C. (2015); Membrane computing to model feature selection of microarray cancer data, *Proceedings of the ASE BigData & SocialInformatics*, 2015.

[10] Garcia-Quismondo, M.; Perez-Jimenez, M. J. Implementing ENPS by Means of GPUs for AI Applications, *Proc. Beyond AI: Interdisciplinary Aspects of Artificial Intelligence*, 27-33, 2011.

[11] Gheorghe, M.; Ceterchi, R.; Ipate, F.; Konur, S.; Lefticaru, R. (2018); Kernel P systems: from modelling to verification and testing, *Theoretical Computer Science*, 724, 45-60, 2018.

[12] Gheorghe, M.; Ipate, F.; Dragomir, C.; Mierla, L.; Valencia-Cabrera, L.; Garcia-Quismondo, M.; Perez-Jimenez, M. J. (2013); Kernel P Systems-Version I, Membrane Computing, *Eleventh Brainstorming Week*, BWMC, 97-124, 2013.

[13] Guzzi, P. H.; Agapito, G.; Cannataro, M. (2014); coreSNP: Parallel processing of microarray data, *IEEE Transactions on Computers*, 63(12), 2961-2974, 2014.

[14] Kentzoglanakis, K.; Poole, M. (2012); A swarm intelligence framework for reconstructing gene networks: searching for biologically plausible architectures, *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 9(2), 358-371, 2012.

[15] Li, J.-M.; Wang, X.-J.; He, R.-S.; Chi, Z.-X. (2007); An efficient fine-grained parallel genetic algorithm based on GPU-accelerated, *Network and parallel computing workshops, 2007, NPC workshops, IFIP international conference on*, 855-862, 2007.

[16] Liu, J.; Iba, H.; Ishizuka, M. (2001); Selecting informative genes with parallel genetic algorithms in tissue classification, *Genome Informatics*, 12, 14-23, 2009.

[17] Maroosi, A.; Muniyandi, R. C. (2013); Accelerated simulation of membrane computing to solve the n-queens problem on multi-core, *International Conference on Swarm, Evolutionary, and Memetic Computing*, 257-267, 2013.

[18] Maroosi, A.; Muniyandi, R. C. (2013); Membrane computing inspired genetic, *Journal of Computer Science*, 9(2), 264-270, 2013.

[19] Mussi, L.; Daolio, F.; Cagnoni, S. (2011); Evaluation of parallel particle swarm optimization algorithms within the CUDA(TM) architecture, *Information Sciences*, 181(20), 4642-4657, 2011.

[20] Nobile, M.; Besozzi, D.; Cazzaniga, P.; Mauri, G.; Pescini, D. (2012); A GPU-based multi-swarm PSO method for parameter estimation in stochastic biological systems exploiting discrete-time target series, *Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics*, 74-85, 2012.

[21] Nobile, M. S.; Besozzi, D., Cazzaniga; P., Pescini, D.; Mauri, G. (2013); Reverse engineering of kinetic reaction networks by means of Cartesian Genetic Programming and Particle Swarm Optimization, *Evolutionary Computation (CEC), 2013 IEEE Congress on*, 1594-1601, 2013.

[22] Pospichal, P.; Jaros, J.; Schwarz, J. (2010); Parallel genetic algorithm on the cuda architecture, *Applications of Evolutionary Computation*, 442-451, 2010.

[23] Sarkar, B. K.; Sana, S. S.; Chaudhuri, K. (2011); Selecting informative rules with parallel genetic algorithm in classification problem, *Applied Mathematics and Computation*, 218(7), 3247-3264, 2011.

[24] Slavik, M.; Zhu, X.; Mahgoub, I.; Shoaib, M. (2009); Parallel Selection of Informative Genes for Classification, *Bioinformatics and Computational Biology. Lecture Notes in Computer Science*, 5462, 388-399, 2009.

[25] Van Nguyen, D. K.; Gioiosa, G. (2010); A region-oriented hardware implementation for membrane computing applications, *Membrane Computing. WMC 2009. Lecture Notes in Computer Science*, 5957, 385-409, 2009.

[26] Zhang, G.; Perez-Jimenez, M. J.; Gheorghe, M. (2017), *Real-life applications with membrane computing*, (Vol. 25): Springer, 2017.

[27] Zhang, G.; Cheng, J.; Gheorghe, M.; Meng, Q. (2013), A hybrid approach based on differential evolution and tissue membrane systems for solving constrained manufacturing parameter optimization problems, *Applied Soft Computing*, 13(3), 1528-1542, 2013.

[28] Zhou, Y.; Tan, Y. (2009); GPU-based parallel particle swarm optimization, *Evolutionary Computation, 2009, CEC'09. IEEE Congress on*, 1493-1500, 2009.