Hindawi Publishing Corporation
Journal of Applied Mathematics
Volume 2014, Article ID 307617, 7 pages
http://dx.doi.org/10.1155/2014/307617

Research Article

Implementation of Membrane Algorithms on GPU

Xingyi Zhang,[1] Bangju Wang,[1] Zhuanlian Ding,[1] Jin Tang,[1] and Juanjuan He[2]

[1] *Key Lab of Intelligent Computing and Signal Processing of Ministry of Education, School of Computer Science and Technology, Anhui University, Hefei 230039, China*
[2] *Key Laboratory of Image Processing and Intelligent Control, School of Automation, Huazhong University of Science and Technology, Wuhan, Hubei 430074, China*

Correspondence should be addressed to Jin Tang; ahhftang@gmail.com

Received 7 May 2014; Accepted 25 June 2014; Published 10 July 2014

Academic Editor: Quanke Pan

Membrane algorithms are a new class of parallel algorithms, which attempt to incorporate some components of membrane computing models for designing efficient optimization algorithms, such as the structure of the models and the way of communication between cells. Although the importance of the parallelism of such algorithms has been well recognized, membrane algorithms were usually implemented on the serial computing device central processing unit (CPU), which makes the algorithms unable to work in an efficient way. In this work, we consider the implementation of membrane algorithms on the parallel computing device graphics processing unit (GPU). In such implementation, all cells of membrane algorithms can work simultaneously. Experimental results on two classical intractable problems, the point set matching problem and TSP, show that the GPU implementation of membrane algorithms is much more efficient than CPU implementation in terms of runtime, especially for solving problems with a high complexity.

1. Introduction

Membrane computing is an emergent branch of natural computing initiated by Păun in 2000 [1], with the aim of abstracting innovative computing models or ideas from the living cells and higher order structures of living cells, such as tissues and organs. The obtained models, called *P* systems, are distributed and parallel computing devices. Most variants of *P* systems were proved to be computationally complete (equivalent to Turing machines or other equivalent computing devices; we also say that *P* systems are universal) as number computing devices [2–4], language generators [5, 6], and function computing devices [7, 8]. For general information on this area please refer to the Handbook of Membrane Computing [9], and for the up-to-date information refer to the membrane computing website http://ppage.psystems.eu/.

P systems have been proved to be a rich framework for handling many problems related to computing. Such systems can theoretically solve presumably intractable problems in a feasible time (solving NP-complete problems [10–12] or even PSPACE-complete problems [13]). Actually, *P* systems can also provide some new ideas for designing optimization algorithms to obtain approximate solutions to the intractable problems [14–22]. The optimization algorithms inspired by *P* systems are usually called membrane algorithms (some researchers also call membrane algorithms *P* systems based optimization algorithms). The first membrane algorithm was proposed by Nishida in 2004 [14], where the nested structure and the communication mechanism between cells were brought from *P* systems. Experimental results showed that such an algorithm is effective and efficient for solving the intractable problem, traveling salesman problem (TSP) [14]. Since then, many membrane algorithms have been proposed for solving various optimization problems, such as knapsack problem [15], point set matching problem [16], numerical optimization problem [17], multiobjective optimization problem [18, 19], DNA sequence design problem [20], and many practical problems [21, 22]. The dynamic behavior analysis of membrane algorithms indicated that a membrane algorithm has a stronger capacity of balancing exploration and exploitation than its counterpart evolutionary algorithm in order to

prevent premature convergence that might occur [23]. We should stress that all membrane algorithms mentioned above can work in a parallel way, in the sense that the evolution of each cell can be performed simultaneously. Although the importance of the parallelism of such algorithms has been well recognized, membrane algorithms were usually implemented on the serial computing device CPU, which makes the algorithms unable to work as expected in a more efficient way. To achieve this aim, this paper considers the implementation of membrane algorithms on parallel computing devices.

The graphics processing units (GPUs) are a kind of computing devices with high parallelism on numerical operations, where massively parallel processors can support several thousands of concurrent threads. The computational power of GPUs has turned them into attractive platforms for general-purpose scientific and engineering applications, especially for tackling large scale numerical computing problems [24]. In this work, we will consider the implementation of membrane algorithms on GPU with an attempt to make the membrane algorithms work in a parallel way. Under the GPU implementation of membrane algorithms presented here, the work of a cell of membrane algorithms is achieved by a thread of GPU, by which all the cells of a membrane algorithm can evolve simultaneously through the concurrent threads of GPU. The GPU implementation ensures that membrane algorithms can work in a more efficient way, in the sense that each operation of membrane algorithms is performed in parallel as much as possible. Experimental results on two classical intractable problems, the point set matching problem and TSP, show that the GPU implementation of membrane algorithms is effective and efficient. Compared with the CPU implementation, the GPU implementation of membrane algorithms consumes much less runtime for dealing with the intractable problems, especially for large scale numerical intractable problems. Software with a friendly interface is also developed for GPU implementation of membrane algorithms, which can provide a convenient tool for the users to solve the intractable problems. We should stress that the GPU implementation of P systems also exists, for example, in Sevilla and Spain [25, 26]. A parallel simulator for membrane computing models on GPU, called PMCGPU, can be found in the website [27].

The rest of the paper is organized as follows. In Section 2, we present the GPU implementation procedure of membrane algorithms. Experimental results on the point set matching problem and TSP are presented in Section 3. Section 4 presents software for GPU implementation of membrane algorithms. Finally, conclusions and remarks are presented in Section 5.

2. Implementation of Membrane Algorithms on GPU

In this section, we will present a GPU implementation procedure of membrane algorithms. We first review its CPU implementation procedure.

Let us recall that a membrane algorithm with a nested structure mainly consists of the following four operations:

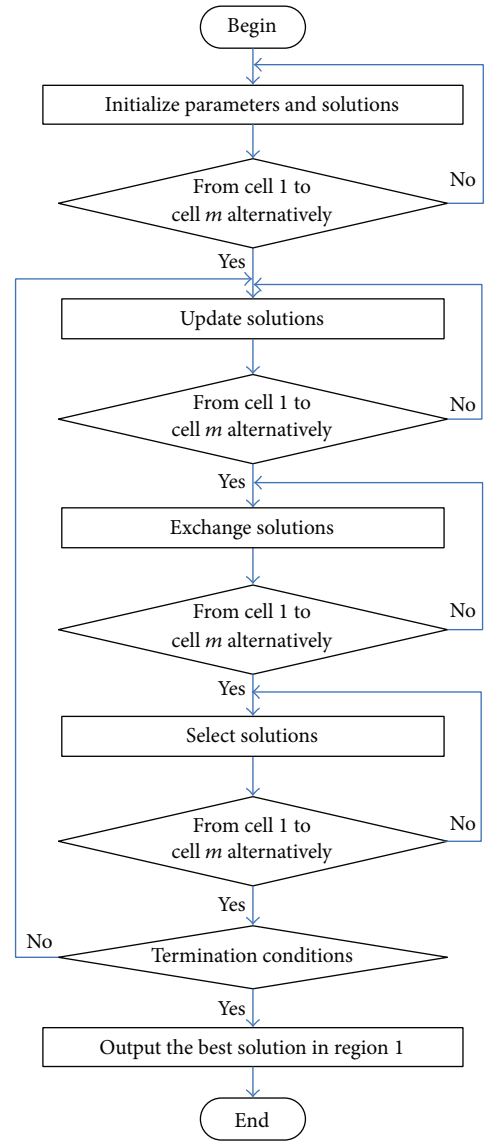

Figure 1: The CPU implementation procedure of membrane algorithms.

(1) initialize solutions in each of the m cells;

(2) update solutions in each of the m cells;

(3) exchange solutions between adjacent cells;

(4) select good solutions in each of the m cells.

Figure 1 illustrates the CPU implementation procedure of membrane algorithms with a nested structure. Under the CPU implementation of membrane algorithms, one cell of the algorithms starts to work only after the work of another cell is finished during the four operations. This means that the cells will evolve one by one during each of the four operations.

Therefore, each of the four operations is achieved in a sequential manner under the CPU implementation. In fact, it is not difficult to find that the four operations can be achieved in a parallel manner, in the sense that the m cells evolve simultaneously if a parallel computing device is used. The

GPU implementation of membrane algorithms can achieve such a parallelism of the four operations.

Briefly, a GPU consists of hundreds of blocks, and each block can support several thousands of concurrent threads. A GPU should work with the help of host CPU. Data can be transferred between threads in the same block through the shared memory in the block. But, the transferred data should be very little due to the small size of the shared memory. Data cannot be directly transferred between threads in different blocks, but only through the host. Figure 2 depicts the structure of a GPU. The proposed GPU implementation procedure of membrane algorithms with a nested structure is shown in Figure 3. Under the GPU implementation presented here, the m cells will work in parallel instead of one by one during the four operations. The parallelism of the m cells is achieved based on an idea as follows: a thread of a GPU does the work of a cell of membrane algorithms, so the m cells can work in parallel through the concurrent threads. This means that GPU has to create the same number of threads as that of cells used by membrane algorithms during the implementation on GPU. Note that, in the GPU implementation procedure of membrane algorithms, a synchronization function has been used after each of the four operations. Under the implementation on GPU, the work of each thread may not be finished at the same time, so this function will ensure that the next operation of membrane algorithms starts only after the work of each of the m cells is finished for the four operations. The parallel work of the cells under the implementation on GPU enables membrane algorithms to work in a more efficient way in terms of runtime, which will be illustrated by simulation experiments in the following section.

3. Experimental Results and Analysis

In this section, we will evaluate the performance of GPU implementation of membrane algorithms through two classical intractable problems: the point set matching problem and TSP. It is useful for readers to have some familiarity with such two problems, so we here briefly recall them.

A point set matching problem can be formulated as follows. Given two sets $P = \{p_1, \ldots, p_m\}$ and $Q = \{q_1, \ldots, q_n\}$, find a map $f : P' \to Q'$, $(P' \subseteq P, Q' \subseteq Q)$ such that the following matching

$$\text{Gobj} = \sum_{p_i, p_j \in P} \left\{ \left| d\left(p_i, p_j\right) - d\left(f\left(p_i\right), f\left(p_j\right)\right) \right| + k\left(n_p - n_{p'}\right) \right\} \quad (1)$$

objective value Gobj is minimized, where $d(p, q)$ is the Euclidean distance between points p and q; n_p, $n_{p'}$ are the sizes of P and P'; and k is a penalty factor.

The TSP raises the following question. Given a list of n cities p_i, $1 \le i \le n$, find a route $p_{i_1} p_{i_2} \cdots p_{i_n} p_{i_1}$ (i.e., this route visits each city exactly once and returns to the origin city), such that the total distance Dist of this route is minimized:

$$\text{Dist} = \sum_{k=1}^{n-1} d\left(p_{i_k}, p_{i_{k+1}}\right) + d\left(p_{i_n}, p_{i_1}\right), \quad (2)$$

where $d(p_i, p_j)$ is the Euclidean distance between cities p_i and p_j.

The membrane algorithm proposed in [16] will be adopted to test the performance of GPU implementation. The number of generations is set as 100, the updating number of each cell is 8 in one generation, and the other parameters are the same as those suggested in [16]. Each test is executed 5 times. All simulations reported in this work are conducted on a PC with a 3.40 GHz Intel Core i7-2600K CPU, Windows XP Professional SP3 64 bit operating system, and NVIDIA GeForce GTX 560 Ti graphics card (it uses the GF114 GPU which offers a maximum of 384 cores). Note that the runtime of GPU implementation reported in the following subsections contains all times that implementing a membrane algorithm on GPU takes, including data reorganization, host-to-GPU data transfer, and GPU-to-host data transfer.

3.1. Experiments on the Point Set Matching Problem. In the experiments, the point set matching problem is created as follows: a point set P consisting of n points is generated randomly in the region $\{(x, y) \mid 0 \le x, y \le 256\sqrt{n/10}\}$, and the observed Q is generated by adding the Gaussian noise with a variance $\sigma = 10$ to set P.

Figure 4 presents the runtime of CPU and GPU implementation of the membrane algorithm having 200 cells on the point set matching problem with different sizes of point set. As shown in Figure 4, the GPU implementation of membrane algorithm is effective and efficient. The GPU implementation outperforms the CPU implementation on the point set matching problem, in the sense that it consumes much less runtime than CPU implementation, especially for point set with a large size. The runtime of CPU and that of GPU implementation will both increase as the size of point set increases. The runtime of CPU implementation will dramatically increase with an increment of the size of point set, while the increment on runtime of GPU implementation is quite slight with an increment of the size of point set. Note that membrane algorithm will perform initializing, updating, exchanging, and selecting operations for the point set matching problem in each cell, and the cells used by the membrane algorithm will be performed in parallel under the GPU implementation instead of one by one under the CPU implementation. So, the runtime will not increase dramatically as the size of point set increases.

Table 1 shows the runtime of CPU and GPU implementation of the membrane algorithm with different numbers of cells on the point set matching problem having 200 points. The GPU implementation of membrane algorithm outperforms the CPU implementation, in the sense that it can make membrane algorithm with a large number of cells work in a more efficient way. The runtime of CPU implementation will dramatically increase as the number of cells used by the membrane algorithm increases, while the implementation

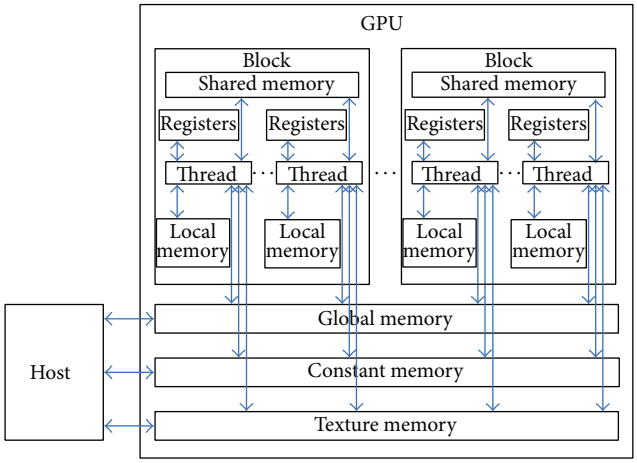

Figure 2: The structure of GPU.

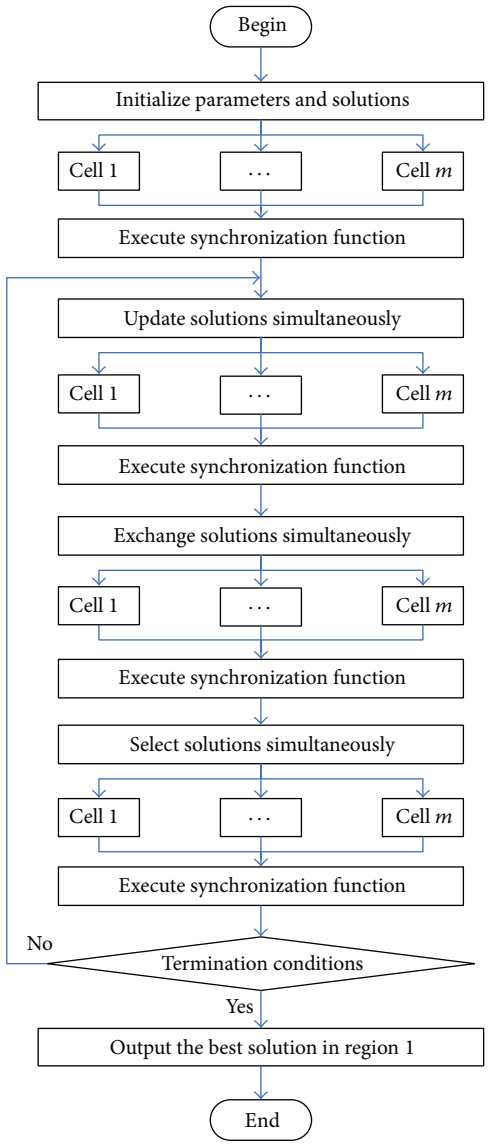

Figure 3: The GPU implementation procedure of membrane algorithms.

Table 1: Runtime (s) of CPU and GPU implementation of the membrane algorithm with different numbers of cells on point set matching problem having 200 points.

Number of cells	Platform	
	CPU	GPU
100	1078.44	103.88
150	1597.06	104.70
200	2082.39	105.30
250	2689.92	106.55
300	3202.36	107.41
350	3599.05	109.38
400	4156.70	111.13
450	4471.67	115.64
500	4988.42	115.66

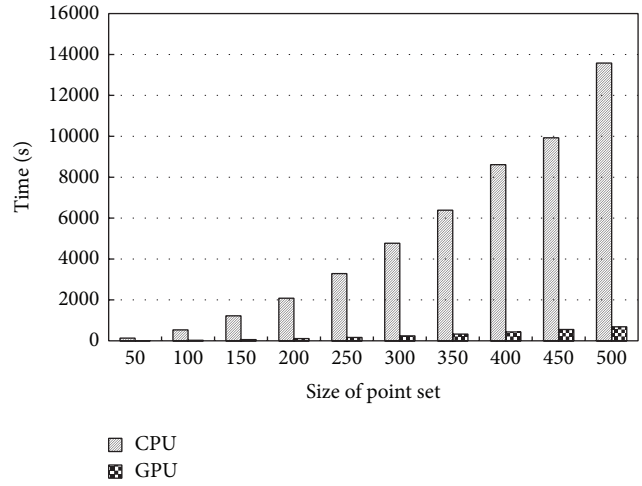

Figure 4: Runtime (s) of CPU and GPU implementation of the membrane algorithm having 200 cells on the point set matching problem with different sizes of point set.

on GPU almost has the same runtime as the number of cells increases. The slight increment on runtime of GPU implementation is partially caused by the increasing overhead generated by the control of the synchronization of cells as the number of cells increases.

3.2. Experiments on the TSP. In the experiments, each TSP is chosen from the TSP benchmark problems in TSPLIB proposed by Reinelt [28]. These benchmark problems have been widely used for testing the performance of an optimization algorithm.

Figure 5 shows the runtime of CPU and GPU implementation of the membrane algorithm having 200 cells on the TSP with different numbers of cities. It is not difficult to find that on TSP there is a similar result as that of the point set matching problem. The GPU implementation of membrane algorithms outperforms the CPU implementation on the TSP in terms of runtime. The GPU implementation will consume much less runtime than CPU implementation, especially on TSP with a large number of cities.

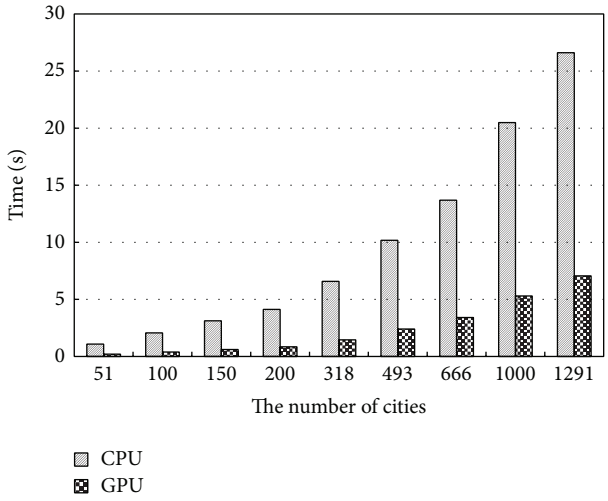

Figure 5: Runtime (s) of CPU and GPU implementation of the membrane algorithm having 200 cells on TSP with different numbers of cities.

Table 2: Runtime (s) of CPU and GPU implementation of the membrane algorithm with different numbers of cells on TSP benchmark problem with 493 cities.

Number of cells	Platform	
	CPU	GPU
100	5.13	2.24
150	7.66	2.34
200	10.17	2.41
250	12.75	2.50
300	15.31	2.56
350	17.75	2.63
400	20.44	2.69
450	22.83	2.77
500	25.56	2.83

Table 2 shows the runtime of CPU and GPU implementation of the membrane algorithm with different numbers of cells on the TSP benchmark problem with 493 cities. As shown in Table 2, on TSP the runtime of CPU implementation of membrane algorithms will increase with an increment of the number of cells, while the implementation on GPU will almost consume the same runtime as the number of cells increases. Different from the case of point set matching problems, the runtime of CPU implementation of membrane algorithms on TSP does not dramatically increase with an increment of the number of cells. This result is mainly caused by the fact that the time consumed by each cell is quite little since the operations in each cell hold a low complexity for the TSP with 200 cities. So, the total runtime of CPU implementation will only increase slightly with an increment of the number of cells.

Compared with the simulation results on the point set matching problem, we can find that the GPU implementation of membrane algorithms is quite efficient for tackling large scale intractable problems, while it is not good at dealing with intractable problems with a small size. The reason for this is that the cells of membrane algorithms working in parallel can save only a little runtime under GPU implementation, since the computational complexity in each cell is very low for intractable problems with a small size. At the same time, under GPU implementation the data should be transferred repeatedly between GPU and host, which will consume some additional runtime. Therefore, the GPU implementation of membrane algorithms will not work in a more efficient way than CPU implementation for solving intractable problems with a small size.

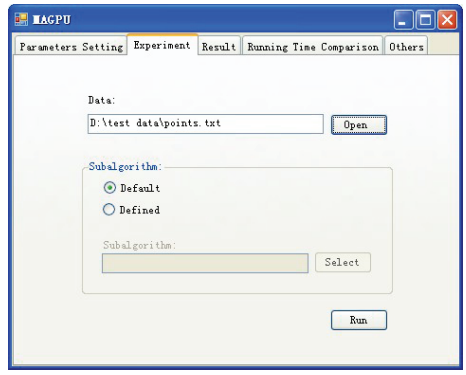

Figure 6: An interface of the software for GPU implementation of membrane algorithms with a nested structure.

4. Software with a Friendly Interface

For the GPU implementation of membrane algorithms, we have to write a new programming code for different intractable problems, which is quite inconvenient for the users. Therefore, software with a friendly interface (termed MAGPU) has been developed, which can provide the users with a convenient tool to implement membrane algorithms with a nested structure on GPU. This software can be found on the website https://github.com/warmheart0/magpu/.

This software mainly contains the following functions: (1) the setting of a few parameters, including the generation number, number of cells, and updating number of each cell in one generation; (2) the choice of experimental data and updating strategy of each cell (it also allows the users to define their own updating strategy); (3) the saving and showing of experimental results; (4) the saving and comparing of runtimes of several experiments; (5) the calculation of a few measurement indexes, such as mean and variance. Figure 6 presents an interface of the software for GPU implementation of membrane algorithms with a nested structure.

5. Conclusions and Remarks

In this paper, the implementation of membrane algorithms on a parallel computing device GPU was carried out. The GPU implementation can achieve the parallelism of membrane algorithms, which makes membrane algorithms work in a more efficient way in terms of runtime. Experimental

results on the point set matching problem and TSP show that the GPU implementation of membrane algorithms outperforms the CPU implementation, in the sense that it consumes much less runtime.

Although the GPU implementation of membrane algorithms has shown a good performance in terms of runtime, many problems remain to be solved for the GPU implementation presented in this work. Among these problems, an interesting one is to further reduce the runtime of GPU implementation of membrane algorithms. In the presented GPU implementation, the data transfer between host and GPU will be performed a large number of times, which takes a lot of runtime. Therefore, a possible solution is to reduce the number of data transfer times between host CPU and GPU. It is conjectured that the runtime of GPU implementation can be greatly reduced by improving the GPU implementation procedure such that only a small number of date transfer times are performed between host and GPU.

In order to provide the users with a convenient tool to implement membrane algorithms on GPU, software with a friendly interface has been developed. This software is only a simple version and many functions need to be improved or added, which is an interesting work that deserves to be further investigated.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (61033003, 91130034, 61272152, 61202011, and 61320106005) and the Natural Science Foundation of Anhui Higher Education Institutions of China (KJ2012A010 and KJ2013A007).

References

[1] G. Păun, "Computing with membranes," *Journal of Computer and System Sciences*, vol. 61, no. 1, pp. 108–143, 2000.

[2] L. Pan and G. Păun, "Spiking neural P systems with anti-spikes," *International Journal of Computers, Communications and Control*, vol. 4, no. 3, pp. 273–282, 2009.

[3] L. Pan and G. Păun, "Spiking neural P systems: an improved normal form," *Theoretical Computer Science*, vol. 411, no. 6, pp. 906–918, 2010.

[4] J. Wang, H. J. Hoogeboom, L. Pan, G. Păun, and M. J. Pérez-Jiménez, "Spiking neural P systems with weights," *Neural Computation*, vol. 22, no. 10, pp. 2615–2646, 2010.

[5] X. Zhang, X. Zeng, and L. Pan, "On string languages generated by spiking neural P systems with exhaustive use of rules," *Natural Computing*, vol. 7, no. 4, pp. 535–549, 2008.

[6] H. Chen, M. Ionescu, T. Ishdorj, and G. Păun, "Spiking neural P systems with extended rules: universality and languages," *Natural Computing*, vol. 7, no. 2, pp. 147–166, 2008.

[7] A. Păun and G. Păun, "Small universal spiking neural P systems," *BioSystems*, vol. 90, no. 1, pp. 48–60, 2007.

[8] X. Zhang, X. Zeng, and L. Pan, "Smaller universal spiking neural P systems," *Fundamenta Informaticae*, vol. 87, no. 1, pp. 117–136, 2008.

[9] G. Păun, G. Rozenberg, and A. Salomaa, *Handbook of Membrane Computing*, Oxford University Press, Oxford, UK, 2010.

[10] X. Zhang, S. Wang, Y. Niu, and L. Pan, "Tissue P systems with cell separation: Attacking the partition problem," *Science China Information Sciences*, vol. 54, no. 2, pp. 293–304, 2011.

[11] T. Song, X. Wang, and H. Zheng, "Time-free solution to Hamilton path problems using P systems with d-division," *Journal of Applied Mathematics*, vol. 2013, Article ID 975798, 7 pages, 2013.

[12] X. Liu, Z. Li, J. Suo, Y. a. . Ju, and X. Zeng, "Solving multidimensional 0-1 knapsack problem with time-free tissue P systems," *Journal of Applied Mathematics*, vol. 2014, Article ID 372768, 6 pages, 2014.

[13] T. Ishdorj, A. Leporati, L. Pan, X. Zeng, and X. Zhang, "Deterministic solutions to QSAT and Q3SAT by spiking neural P systems with pre-computed resources," *Theoretical Computer Science*, vol. 411, no. 25, pp. 2345–2358, 2010.

[14] T. Y. Nishida, "An approximate algorithm for NP-complete optimization problems exploiting P systems," in *Proceedings of the Brainstorming Workshop on Uncertainty in Membrane Computing*, pp. 185–192, Palma, Spain, 2004.

[15] G. Zhang, M. Gheorghe, and C. Wu, "A quantum-inspired evolutionary algorithm based on P systems for knapsack problem," *Fundamenta Informaticae*, vol. 87, no. 1, pp. 93–116, 2008.

[16] J. Tang, Z. Ding, X. Zhang, and B. Luo, "Membrane computing model based algorithm for point set matching," *Infrared and Laser Engineering*, vol. 42, no. 5, pp. 1388–1394, 2013.

[17] J. Cheng, G. Zhang, and X. Zeng, "A novel membrane algorithm based on differential evolution for numerical optimization," *International Journal of Unconventional Computing*, vol. 7, no. 3, pp. 159–183, 2011.

[18] H. Liang, R. He, N. Wang, and Y. Xie, "P systems based multi-objective optimization algorithm," *Progress in Natural Science. English Edition*, vol. 17, no. 4, pp. 458–465, 2007.

[19] L. Huang, I. H. Suh, and A. Abraham, "Dynamic multi-objective optimization based on membrane computing for control of time-varying unstable plants," *Information Sciences*, vol. 181, no. 11, pp. 2370–2391, 2011.

[20] J. H. Xiao, X. Y. Zhang, and J. Xu, "A membrane evolutionary algorithm for DNA sequence design in DNA computing," *Chinese Science Bulletin*, vol. 57, no. 6, pp. 698–706, 2012.

[21] G. Zhang, C. Liu, and H. Rong, "Analyzing radar emitter signals with membrane algorithms," *Mathematical and Computer Modelling*, vol. 52, no. 11-12, pp. 1997–2010, 2010.

[22] G. Zhang, J. Cheng, M. Gheorghe, and Q. Meng, "A hybrid approach based on differential evolution and tissue membrane systems for solving constrained manufacturing parameter optimization problems," *Applied Soft Computing Journal*, vol. 13, no. 3, pp. 1528–1542, 2013.

[23] G. Zhang, J. Cheng, and M. Gheorghe, "Dynamic behavior analysis of membrane-inspired evolutionary algorithms," *International Journal of Computers, Communications & Control*, vol. 9, no. 9, pp. 227–242, 2014.

[24] A. Ruiz, M. Ujaldón, J. A. Andrades et al., "The GPU on biomedical image processing for color and phenotype analysis," in *Proceedings of the 7th IEEE International Conference on Bioinformatics and Bioengineering*, pp. 1124–1128, New York, NY, USA, January 2007.

[25] F. G. C. Cabarle, H. N. Adorna, M. A. Martinez-Del-Amor, and M. J. Pérez-Jiménez, "Improving GPU simulations of spiking neural P systems," *Romanian Journal of Information Science and Technology*, vol. 15, no. 1, pp. 5–20, 2012.

[26] J. M. Cecilia, J. M. García, G. D. Guerrero, M. A. Martínez-del-Amor, I. Pérez-Hurtado, and M. J. Pérez-Jiménez, "Simulating a P system based efficient solution to SAT by using GPUs," *The Journal of Logic and Algebraic Programming*, vol. 79, no. 6, pp. 317–325, 2010.

[27] The parallel simulators PMCGPU for membrane computing on the GPU, http://sourceforge.net/projects/pmcgpu/.

[28] G. Reinelt, TSPLIB, http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/tsp/.