A Novel Parallel Clustering Algorithm Based on Artificial Immune Network Using nVidia CUDA Framework

Ruiyi Luo and Qian Yin*

Image Processing and Pattern Recognition Laboratory
Beijing Normal University, Beijing 100875, China
yinqian@bnu.edu.cn

Abstract. In this paper, a novel parallel data clustering algorithm based on artificial immune network aiNet is proposed to improve its efficiency. In consideration of the restrictions of GPU, we carefully designed the data structure, algorithm flow and memory allocation strategy of the parallel algorithm and realized it using NVIDIA's CUDA framework. During the implementation, in order to fully explore its implicit parallelism, we allocated threads on GPU that represent the network cells of aiNet, and modified this algorithm to let those thread operations parallel during the clustering process. We calculated the affinity parallel, combined the random numbers with the local search algorithm to select the first n cell parallel, and did the network suppression parallel. Experimental results show that certain speedup can be obtained by using the proposed method.

Keywords: artificial immune network, aiNet, clustering, parallel, GPU.

1 Introduction

Artificial immune systems have drawn much attention in recent years. Many interesting algorithmic solutions were proposed in some fields, such as clustering or classification [1][2] in data mining, intrusion detection of network [3], optimization [4] and etc. Clustering is an unsupervised classification, which is very useful for data analysis [5][6]. The most popular algorithm for data clustering is K-Means [7]. Though the speed of K-Means is usually very fast, a main drawback of it is that it needs to know the cluster number previously and the result is sensitive to the initial center. In contrast, the artificial immune network clustering is a dynamic clustering algorithm, which means that it can get the clustering number in the analysis process. However, the efficiency of it is lower, and this is because that there are too many calculation steps during the process. The objective of our study is to parallel the aiNet (one of artificial immune network clustering) to improve its efficiency. As we known, the biological immune system is obviously a parallel system to protect the vertebrate's organisms, that's to say, the algorithm that simulates it has inherent parallelization and can be explored. Graphic Processing Unit (GPU) is a highly parallel multithread and multi-core

* Corresponding author.

J.A. Jacko (Ed.): Human-Computer Interaction, Part I, HCII 2011, LNCS 6761, pp. 598–607, 2011.

processor originally designed for accelerating 3D rendering in graphic applications, where need paralleling data processing for speed. It has emerging as one of the most powerful and common parallel processing devices, we also choose it as our parallel tool.

The paper is organized as follows. In section 2, artificial immune network algorithm used in data clustering is introduced, and time complexity analysis is also given. In section 3, we describe the relevant architectural features of GPU and Compute Unified Device Architecture (CUDA) framework. The Implementation of the parallel algorithms using CUDA framework on GPU is proposed. Section 4 shows the empirical results obtained. Section 5 concludes this paper and gives some guidelines for the future work.

2 The aiNet Algorithm

Since the immune network theory had been proposed by Jerne in 1974 [8] and the colonel selection and affinity maturation algorithms proposed by Burent in 1987 [9], there are several useful principles which can be simulated in engineering research were emerged in human immune system. The two main principles of which are as follows. One is the colonel selection principle [10], which explains how the immune system recognizes and reacts to antigen(Ag). The second is the immune network theory[8], which hypothesis a new view point of lymphocyte activities, memory cells, internal images, antigen, antibody(Ab), size control, affinity, etc. In this paper, we choose the aiNet [1] algorithm to parallel for data clustering, which is based on the theory of artificial immune network .

The aiNet algorithm was proposed by De Castro in 2000 and has been described detailed in reference [1]. However, in order to find out which part of this algorithm could be paralleled to improve its efficiency, we find out the most time-consuming processes of the aiNet algorithm and try to execute them parallel. Firstly, In each iteration process, all network cells do the same thing to the antigen presented when calculate affinity, so the time consumed in those steps should be the time one network cell consumed multiply by the size of the network. Secondly, in the selection process, the time of selecting the first N cells takes no less than $o(N * \log(N))$. Thirdly, in network suppression process or clone suppression process, similarity between memory cells is measured, and this step takes no less than$o(N_Size^2)$ (N_Size is the size of the network), which are very time-consuming especially when the N_Size is rather large.

3 The Parallel Algorithm

3.1 Data Structure and the Flowchart

Parallel technology has been widely used in various algorithms to improve the efficiency [11][12][13]. GPU is a highly parallel multithread and multiprocessors which is excellent at floating point operation and parallel computing. It is very appropriate to solve problems that can be expressed as data-parallel, as the same instructions are executed for each data element. There are some other algorithms have been paralleled by researchers, such as the clone selection [14], resource limited

artificial immune system [15], and Genetic Algorithm have been carried out on GPU [16]. However, those algorithms are mainly based on the colonel selection principle [10], and our work in this paper is to parallel the immune network system for data clustering based on immune network theory [8].

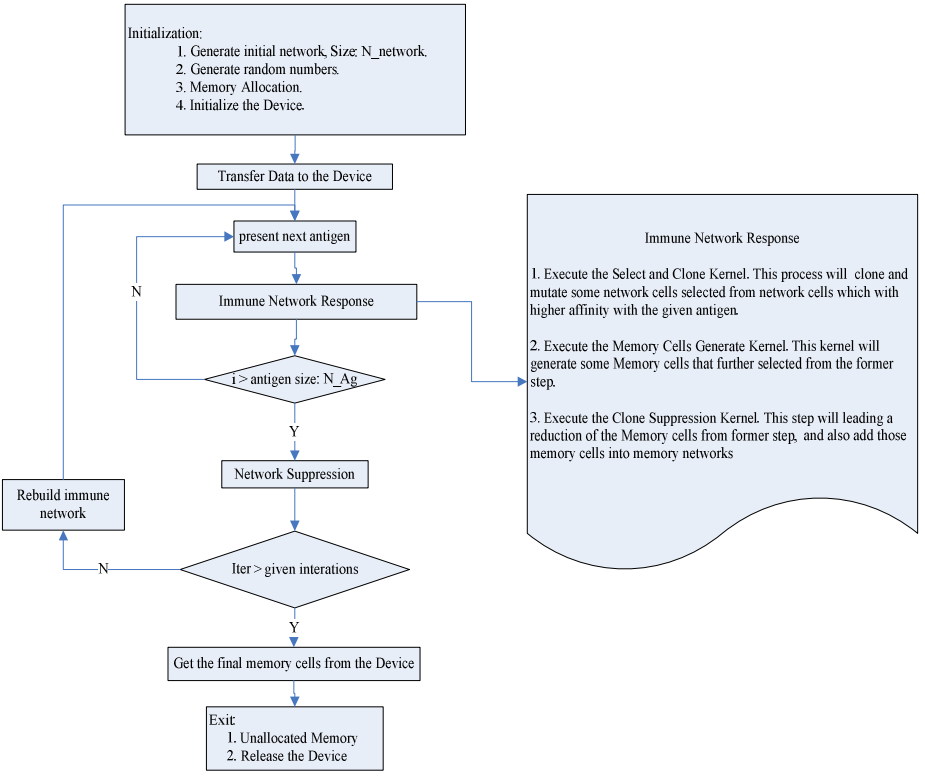

Fig. 1. Flow chart of the proposed algorithm

To implement our parallel algorithm on GPU, we first design the data structure, which takes shape logically in form of a two-dimensional grid plane and is allocated linear storage in shared memory of GPU. A grid of it represents a cell and has four fields: Ab, Affinity, Is_Deleted, Is_HillClimbing. The Ab represents the vector of length n, which is the measurement of the antigen, and it represents the data information that needs for clustering in the real world. The Affinity is a float data which store the affinity value of the cell with the other cells. Is_Deleted indicates whether this cell has been deleted from the network. Is_HillClimbing represents whether we will execute hill climbing process with the cell. More details will be introduced in the following sections. Firstly, we distribute network cells on shared memory and allocate memory on GPU. Then, in each iteration process, for the antigen presents serially to the network, we start the immune response where threads are running parallel. Finally, the network will be rebuilt and the iteration process will be continued. The whole algorithm we proposed briefly works as Fig. 1.

We use the random numbers for the data initialization which are generated by a Kernel introduced in 3.4, and the immune network response process which contains two other kernels will be presented in 3.2 in detail.

3.2 Immune Network Response

The general flow has been introduced in the last section. In this section, more details of immune network response are introduced, which are shown in Fig.2.

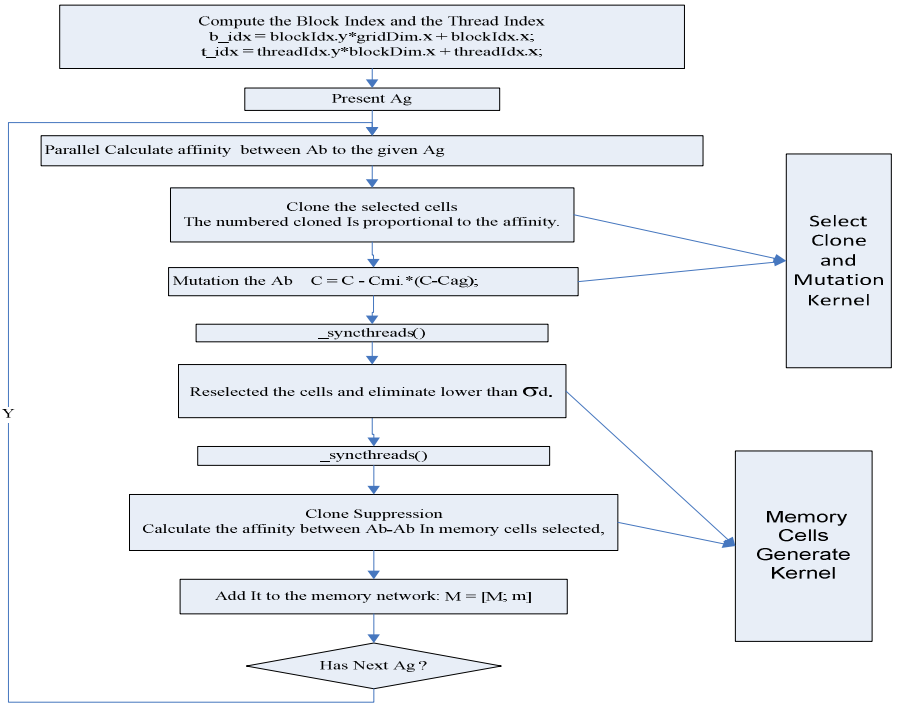

Fig. 2. Immune Response Flowchart

Corresponding to the Fig.2, for the antigen present serially, algorithm we proposed works as follows in detail.

- carry out the step 1 and step 2.
    - Step1: Execute the Select and Clone Kernel: calculate its affinity to all the network cells and select the n highest ones, then clone and mutate them. The higher the affinity, the larger the clone size is.
    - Step2: Carry out Memory Cells Generate Kernel, which recalculates the affinity of the selected cells by step1 with the antigen presented and eliminate those inferior to the threshold δ_d. And reselect the ξ% most improved cells sorted by affinity. This is also being executed on GPU parallel. Then, execute the clone suppression during which eliminate

those cells whose affinity (Ab-Ab) is inferior to threshold. The memory cells generated are put in memory network.

- Combine the memory cells with the network cells, and execute the Network Suppression Kernel: calculate the whole network inter-cell affinities and eliminate those cells whose affinity with each other is inferior to a given thresholdδ_s; Replace the whose individuals by novel randomly generated ones;

Among the implementation, one of the most challenges is the memory allocation. Firstly, memories are allocated with the size of the network cells on shared memory of GPU to initialize our algorithm; also we allocate some more memories for amplification as after the outer iteration the size of the network will increase. However, the max size is the sum of the initial network size added with the size of Memory Network Cell. For the Clone Network, the max size is equal to the Clone Number Multiplier (a pre-set number) multiplied by the number of threads selected to execute the hill climbing algorithm.

There is also a Network Suppression Kernel like the Clone Suppression Kernel, which is executed after the antigens all have been represented once. For each antigen, the whole network response to it parallel, each thread calculates the affinity, then, judged by itself whether started the local search, and the Is_HillClimbing is set before current iteration started. Euclidean distance is used to calculate the affinity between Ag and Ab or between Ab and Ab.

3.3 Generation of Random Number

The CUDA haven't provided any Random number generator. Here we use the example named MersenneTwister provided by the SDK of CUDA, and the example is based on the algorithm proposed by Makoto Matsumoto and Takuji Nishimura. We need random numbers in the following process: Firstly, we initialize the network cells with the random numbers, which is equal with: N_Network_Size (the size of the network). Secondly, when we decide which thread will continue with the hill climbing local search, some random numbers are also needed, which are transformed into the subscript of the network cells and thread.

$$\xi * \sum_{i}^{N\_Selected} affinity_i * CloneMultiple. \quad (1)$$

What's more, we need to carry out the local search twice during the total process. In the first search, the numbers we need is N_Selected (a pre-set number). In the second search, the numbers we need is calculated as formula (1). We select $\xi\%$ higher affinity cells and clone them. The number of clone size is equal with the value of affinity multiple by CloneMultiple (a pre-set value). Thirdly, when we rebuild the network, we need random numbers to replace the weakest ones in every interaction, and the number of random numbers we needed are N_Network_Size.

The total max number of random we need is as formula (2).

As the $affinity_i$ of each selected cells is not same, it cause that the N_{Random} is not a certain data. However, we use $affinity_{max}$ which is the max affinity value among those cells instead of different $affinity_i$ to get a certain N_{Random}. This generated process is executed once at the start of iteration.

$$N_{Random} = N_{Network_{Size}} + N_{Selected} + \xi\% * \sum_{i}^{N\_Selected} affinity_i * CloneMultiple + N\_Network\_Size. \quad (2)$$

3.4 Parallel Local Search

In the proposed algorithm, we use the hill climbing to search the first n affinity cells. To make the program continuous and avoid switch threads, we use random numbers to decide which thread will execute this process. If the thread starts, it firstly selects the one with the highest affinity from its four sides in the data structure: left, right, up, down. Then if the affinity of the center cell where the thread start is larger than the other four, the thread will terminate. Otherwise, the highest one will be the center and continue the same operation. In this process, in order to avoid that all the threads get one highest cell but the n (N_Selected) first, the number of iterations (ITER_HillClimbing) is set a certain value based on the ratio of n and total network cells number. The left part of Fig.3 has shown a thread that carries out local search of the hill climbing. If the ITER_HillClimbing we set was small, it would only search for a smaller scope, such as in our grid data structure. If we set ITER_HillClimbing to three, most eleven grids we could search are around the center blue cell. As the right part of Fig.3 shown, the empty grids represent there are no cells generated or have been deleted, we will take c as the below cells of A. If we first get a center with B, we will also search its four sides, and then we get b in the first iteration, and then start the next iteration from the center of b.

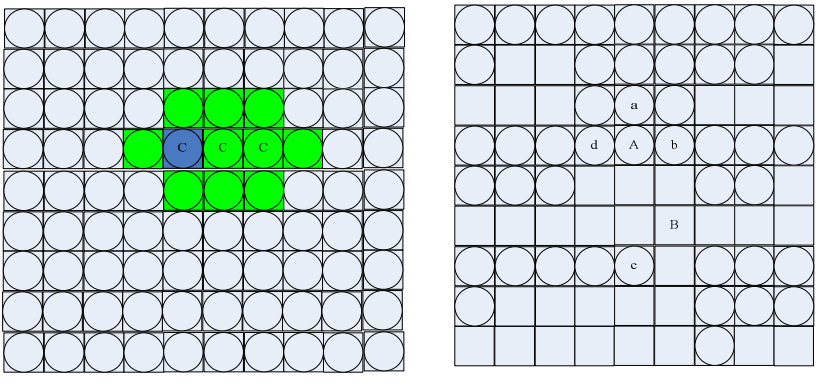

Fig. 3. The instruction of the Hill Climbing Local Search

In our implementation, after we selected the first n cells using the proposed methods in Clone and Mutation Kernel, the clone and mutation would start. In Memory Cells Local Search Kernel of our algorithm, we follow it by the implementation of eliminating those cells whose affinity is inferior to the given threshold.

3.5 Clone Suppression and Network Suppression

In the two suppression processes (clone suppression and network suppression), we need to calculate the affinity between the network cells. Based on the data structure we proposed, the calculation of Ab to all other Ab started. In this progress, we just need to judge if it satisfied the certain threshold and needn't save the calculation result. If it is inferior to threshold, the Is_Deleted flag is set to one, and then stop the calculation with the remaining cells, which means that it has been deleted from the network. The time complexity of this method is o (n), because one Ab need to compute with all other network cells once. We provide the flowchart in Fig.4, which has some detailed design as follows. The program judged the Is_Deleted at first to avoid the unnecessary calculation. If the Is_Deleted attribute has been set to one, this calculation will be canceled. The setting of this flag uses the atomic operation, that is to say, the data can be operated only by one thread at a time. Therefore, this mechanism also insures that we will not delete both the two Abs whose affinity between them is below the threshold.

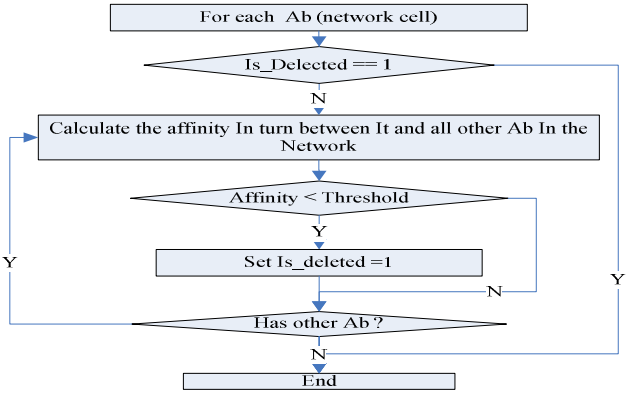

Fig. 4. The Flowchart of the Suppression

4 Experimental Results

A PC was used for execution and performance analysis: AIUS with Intel CPU i5 480 2.63 GHz processor, NVIDIA GeForce GT 425M. The applications presented in this paper were implemented on it. The CUDA timer functions are used to compute the time consumed by each operation. The speedup was evaluated in two experiments. The objective of first experiment is to test the speedup of different scales of records. And the second experiment shows that whether different number of antibodies influence the speedup.

In the first experiment, the data sets had 1000, 10,000, 100,000 records respectively, and each record set has three subsets, in which the record has two, four and eight attributes respectively. The antibody of our network is set to be 512 threads in one grid. From the Fig.5, we can see speedups for different number of clusters respectively of our result at the given number of threads. This speedup we got is due to that all the network

cells parallel response with the antigen. From the Fig.5, we can see that the speedup increased with the data record size increased. However, due to that the antigens are present to the network serially, the speedup is not increased the same as the multiple of the data record increased. Also, we can see from the Fig.5 that the speedup is increased as the attributes increased, which is because that the affinity calculation processes are all modified to be parallel.

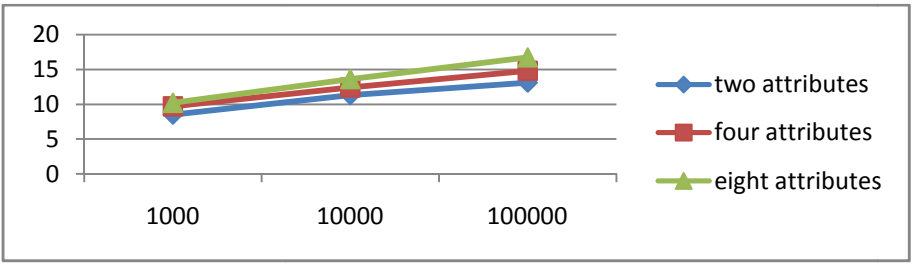

Fig. 5. Speedup with different size of data records

In the second experiment, there have 10000 records, and also have two, four, eight attributes respectively. However, the network size is given to 64, 256, 512, and 1024. As showed in Fig.6, all of them are tested under 1,000 records. With more network cells, more threads are started. However, speedup went up not so obviously, that is to say, the increase of network cells has little influence on speedup. Which is because that after the first selection process of network, the following steps are calculated on the first n cells selected. Therefore, there are no much differences achieved when change the network cells size.

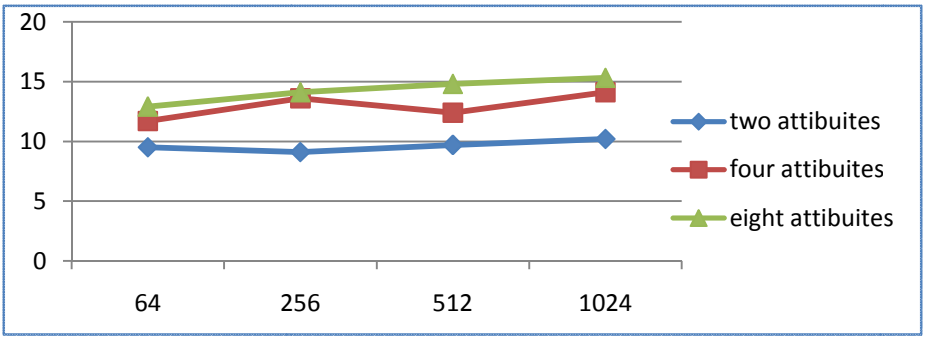

Fig. 6. Speedup with different network cells size

5 Conclusion

In this paper, parallelization of immune network based on the structure of GPU was investigated to improve the efficiency and scalability of AIN for data clustering. Firstly, we analyzed the time complexity of the aiNet algorithm and found that some

parallel work could be done to improve the efficiency. We implemented our parallel immune network algorithm on GPU. The three kernels we had designed in section 3 executed on GPU to do the main task of data clustering. During the whole process, some skills were designed because we tried to make sure that the thread executed frequently to get higher speedup. For example, we combined the random numbers with the hill climbing algorithm as our local search method for the first n higher affinity cells; we carried out the network suppression in parallel introduced in section 3.6; we generated enough random numbers before the iteration process to make our algorithm high efficiency. The experiment result shows that the algorithm proposed have achieved about 10 times speedup than the original algorithm with the parallel technology.

It is noticed that the flow of the algorithm is relative complexity. Therefore, in the future work, we will try to optimize memory allocation and simplify the algorithm to make it more proper for practical using.

Acknowledgments. The research work described in this paper was fully supported by the grants from the National Natural Science Foundation of China (Project No. 90820010, 60911130513). Prof. Qian Yin are the author to whom all correspondences should be addressed.

References

1. De Castro, L.N., Von Zuben, F.J.: An evolutionary immune network for data clustering. In: Proc. Sixth Brazilian Symp. Neural Networks, pp. 84–89 (2000)
2. Timmis, J.: Artificial Immune Systems: A Novel Data Analysis Technique Inspired by the Immune Network Theory. Ph.D. Dissertation, Department of Computer Science, University of Wales (2000)
3. Thomas, S., Jonathan, T., Claudia, E.: A Comparative Study of Real-Valued Negative Selection to Statistical Anomaly Detection Techniques. In: Jacob, C., Pilat, M.L., Bentley, P.J., Timmis, J.I. (eds.) ICARIS 2005. LNCS, vol. 3627, pp. 262–275. Springer, Heidelberg (2005)
4. De Castro, L.N., Timmis, J.: An artificial immune network for multimodal function optimization. In: Proc. Congr. Evol. Comput., pp. 699–704 (2002)
5. Everitt, B., Landau, S., Leese, M.: Cluster Analysis. In: Everitt, B. (ed.), 4th edn. Hodder Arnold, London (2001)
6. Gao, X.B.: Fuzzy Clustering Analysis and Application. In: Gao, X.B. (ed.), pp. 49–160. Xidian University Press, Xian (2004) (in chinese)
7. Han, J., Kamber, M.: Data Mining: Concepts and Techniques. Morgan Kaufmann, San Fransisco (2000)
8. Jerne, N.K.: Towards a Network Theory of the Immune System. Ann. Immunol. 125C, 373–389 (1974)
9. Burnet, F.M.: Clonal Selection and After. In: Bell, G.I., Perelson, A.S., Pimbley Jr., G.H. (eds.) Theoretical Immunology, pp. 63–85. Marcel Dekker Inc., New York (1978)
10. Ada, G.L., Nossal, G.J.V.: The clonal selection theory. Scient. Am. 257, 50–57 (1987)
11. Hong, P., Wang, M., Lai, C.H.: Design of parallel algorithms for fractal video compression. Int. J. of Comput. Math. 84, 193–202 (2007)

12. Souravlas, S., Roumeliotis, M.: On further reducing the cost of parallel pipelined message broadcasts. Int. J. Comput. Math. 83, 273–286 (2006)
13. Sukumar, M., Madhumangal, P., Tapan, K.P.: Optimal sequential and parallel algorithms to compute a Steinertree on permutation graphs. Int. J. Comput. Math. 80, 939–945 (2003)
14. Watkins, A., Bi, X., Phadke, A.: Parallelizing an immune-inspired algorithm for efficient pattern recognition. In: Intelligent Engineering Systems through Artificial Neural Networks: Smart Engineering System Design: Neural Networks, Fuzzy Logic, Evolutionary Programming, Complex Systems and Artificial Life, vol. 13, pp. 225–230. ASME Press, New York (2003)
15. Watkins, A., Timmis, J.: Exploiting parallelism inherent in AIRS, an artificial immune classifier. In: Nicosia, G., Cutello, V., Bentley, P.J., Timmis, J. (eds.) ICARIS 2004. LNCS, vol. 3239, pp. 427–438. Springer, Heidelberg (2004)
16. Jianming, L., Lihua, Z., Linlin, L.: A Parallel Immune Algorithm Based on Fine-grained Model with GPU-Acceleration. In: 2009 Fourth International Conference on Innovative Computing, Information and Control (2009)