MULTI-AGENT BASED SIMULATION OF LARGE RANDOM BOOLEAN NETWORK

Dang Hai PHAM

Laboratoire de Cognition Humaine et Artificielle, Ecole Pratique des Hautes Etudes
41 Rue Gay Lussac, 75005 Paris, France

haipd-fit@mail.hut.edu.vn

ABSTRACT
Random Boolean networks, a generalization of cellular automata, were originally introduced as a simple model of genetic regulatory networks, but they are also used as mathematical models for studying complex dynamical systems with a large number of coupled variables. Simulating sequentially large networks with high connectivity meets often with difficulties on the time and the memory. We propose here a multi-agent based approach for simulating large random Boolean networks, which promises to give an improvement of its performance by using multiprocessors systems.

Keywords: Random Boolean network, Multi agent based simulation, Synchronization algorithm

1. INTRODUCTION

Random Boolean Networks (RBN), also known as Kauffman network or Nk-network, were originally introduced by Stuart Kauffman (Kaufman 1969) as a simple model to study the dynamics of complex genetic regulatory systems. In a random Boolean network, each gene is represented by a vertex in a directed graph. An edge from one vertex to another implies a causal link between two genes. The ON state of a vertex corresponds to the gene being expressed. Time is discrete, and at any time point, the new state of a vertex v is a Boolean function of the previous states of the vertices which are predecessors of v.

Up to now, the scope of existing analytical result is rather restricted, very few exact solutions have been determined. Instead computer simulations and statistical analysis are used to gain theoretical understanding. There are several tools available for exploration of different properties of RBNs, such as **DDLab** (*http://www.ddlab.com*), **RBN Toolbox** (*http://www.teuscher.ch/rbntoolbox*), **RBNLab** (*http://rbn.sourceforge.net*), but none of them can be used for the network with many vertex and high connectivity. That is because the sequential simulation of the large one meets often with difficulties on the time and the memory.

Large simulation consumes usually enormous amounts of time on sequential machines, while Multi-Agent Based Simulation can distribute its agents over different processors, thus reduces the computation time, even though, it always requests synchronization for agents and depends robustly on the communication between agents. We believe that it will be an appropriate approach for simulating a large random Boolean network.

Having it in mind, we organize the rest of this paper as follows. The classical random Boolean network and the multi-agent based simulations are introduced in the section 2 and section 3 respectively. We propose in the section 4 a multi-agent based modeling random Boolean network. The section 5 gives some experiment results. This paper is closed with the conclusion and future work in section 6.

2. RANDOM BOOLEAN NETWORKS

A Classical RBN is a directed graph consisting of N nodes, where each node is connected to k other (or the same) nodes. The parameter k is known as the *connectivity* of the network. The set of k nodes connected to a given node is referred as its neighbourhood. Each node i is characterized by a Boolean (or binary) value $x_i \in \{0,1\}$ and a Boolean (or binary) function f_i, which assigns to each of the 2^k states of its k input nodes an output 1 with probability p or 0 with probability 1-p. The parameter $p \in [0, 1]$ is known as the *bias* of the logic function. The k connections and the Boolean function of each node are chosen randomly at the beginning but remain fixed during the dynamics of the network

In an RBN model, time is viewed as proceeding in discrete steps. At each step, the state of each node evolves according to its logic function and the previous states of its neighbourhoods. Let $v_i = (x^1_i,\ldots x^k_i)$ be set of k inputs of node x_i, then $x_i(t+1) = f_i(x^1_i(t),\ldots x^k_i(t))$ or $x_i(t+1) = f_i(v_i(t))$. Updating the state of all nodes in classical RBN is done synchronously. If $v(t) = \{v_1(t),.. v_N(t)\}$ is the state vector of the network at time t then evolution equation can be written $v(t+1) = f(v(t))$ with $f(v(t)) = \{f(v_1(t),\ldots f(v_N(t))\}$

Because the number of possible state of a network is finite, i.e. it equals to 2^N, so giving some initial state, evolution of the network in time traces a trajectory through the state space. If, after some period of time, the network comes back to one of the states it already visited L steps in the past, it will forever retrace this sequence of L states. We say that the system has fallen

into a *period L attractor*. Attractors of length L = 1 are called *point attractor* or *fixed points* of the network. The attractor states and the transient states leading to the attractor together constitute the basin of attraction. The number and length of attractors represent two important parameters of the cell modelled by a RBN. The number of attractors corresponds to the number of different cell types. The attractor's length corresponds to the cell cycle time which refers to the amount of time required for a cell to grow and divide into two daughter cells (Kauffman 1993). There are analytic solutions of RBN for k = 1 and k = N (Flyvberg and Kjaer 1988; Derrida and Flyvbjerg 1987), but finding general analytic solution is still an open issue. An example of a RBN with N=3 and k= 2 is shown in Figure 1.

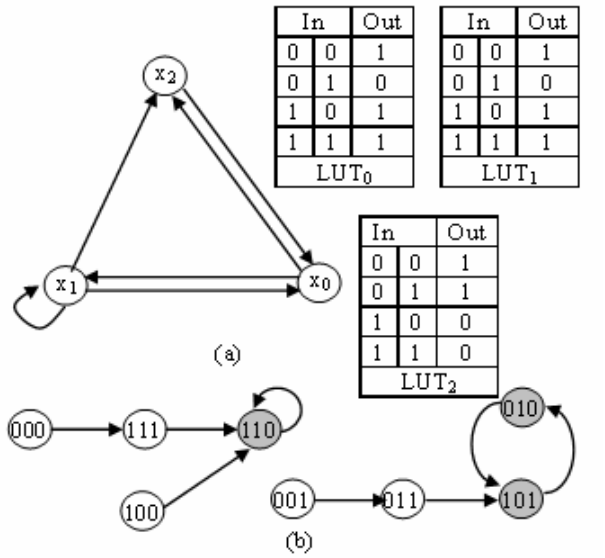

Figure 1: Example of classical RBN with N=3 and k=2 a) Directed graph and Boolean function using lookup table (LUT) for each node. b) State space diagram: there is one point attractor (110) and one cycle attractor of period two (101 010)

The parameters k and p determine the dynamics of the network. For a given bias p, there is a critical line $K_c = 1/2\,p(1-p)$, below which the network is in the *frozen phase* and above which the network is in the *chaotic phase* (Derrida and Pomeau 1986). On the critical line between the frozen and the chaotic phases, the network exhibits *self-organized critical behaviour*, ensuring both stability and evolutionary improvements.

In the classical random Boolean networks, one important aspect is that the networks are assumed to be synchronous; however it is not very realistic as genes do not change their state all at the same moment, hence it is more appropriate to update these systems asynchronously using a random order. The exploration of such asynchronous RBNs can be found in (Harvey and Bossomaier 1997; Di Paolo 2001; Gershenson 2004).

3. MULTI-AGENT BASED SIMULATION

Agent-based simulation models are normally based on a discrete time model, where system states are changed at discrete time points only. The two most common types of such discrete time simulation are called *time-stepped* and *event-driven* (or *event-stepped*) models (Fujimoto 2000). They are distinguished by the mechanism used to advance simulation time. In the first approach, the simulation advances in equal-size time steps. In each step, the clock is adjusted and the participants are informed about the new time. They then can check if an activity has to be executed at this point in time. In the other one, a timestamp is assigned to each event which is an abstraction used in simulation to model some instantaneous action in the real system, to indicate the point in simulation time, when the event occurs. Each event usually results in some change in state variables defined by the model.

In Multi-agent based simulation (MABS), system entities are modelled using multiple agents which interact with the other agents by sending messages to them and to their environment. Agents can be viewed as threads of control, which execute concurrently (Wooldridge 2002). Therefore, it is important that the agents perceive a common view of the virtual environment. For example, any pair of agents should perceive a set of messages in the same sequence, or in other words, each agent has to process events/messages according to their timestamp order. This is called *local causality constraint* (Fujimoto 1990). Two main approaches have been proposed to ensure that the local causality constraint is not violated: *conservative* and *optimistic* synchronization.

The first one avoids strictly the causality violations by processing safe events only. The well-know algorithm **CMB** (Bryant 1977; Chandy and Misra 1979) ensures the local causality constraint is never violated but may lead to deadlock situation. The *null message* scheme (Misra 1986), on its turn, can avoid this deadlock but may generate an excessive number of null message. In addition, this technique relies heavily on the concept of *look-ahead* (Nicol 1996) and also typically requires a static connection between agents.

The optimistic approach, in contrast, allows causality error to occur but provides suitable techniques to recover from an incorrect system state. The best know is **Time Warp** algorithm (Jefferson 1985). It uses a mechanism called *rollback* to cancel the erroneous computation and reprocess messages in their timestamp order whenever a *straggler message* occurs. Several additional strategies of memory management and cancelling incorrect computation to improve efficiency of this algorithm can be found in (Fujimoto 1990; Jefferson 1990; Lin and Preiss 1991). The Time Warp exploits full parallelism of systems, nevertheless it is hard to implement because of requiring complex manipulations.

Another problem in multi-agent systems is the accessing to shared state variables which are maintained by the different agents. This usually leads to an all-to-all

communication between agents and results in the degradation of simulation performance. Many investigations in the context of Multi-Agent Based Simulation have addressed to the avoiding this broadcast communication. Most of them limit the number of agents receiving message based on their location or other application-specific attributes. Algorithms such as can be found in (Macedonia et al. 1995; Logan and Theodoropoulos 2001; Ewald et al. 2006).

4. MODELING RANDOM BOOLEAN NETWORK

We model the Random Boolean Network as a multi agent system. Each node of the network is modeled as an agent that consist of three main fields: agent's state which is a Boolean variable, list of its neighborhoods, which is a dynamic array (because the networks can be non-homogeneous, k is not the same for all agents, such as a scale-free network has an exponential distribution of connectivity, so although most nodes may have a small connectivity, some can have large values) and a lookup table which can be represented as an (array of) integer. The agents behave step by step. At each step in its evolution, an agent reads the state of all its neighborhoods refers to its lookup table and then updates its state.

In the distributed simulation, we can simply model each agent as a process which behaves independently. These agents are distributed over different processors and interacts each other by exchanging messages. At each step, after updating its state, each process send (k) messages to (k) its neighborhoods. Clearly, it is an effectless approach because there are too many of message sent in each step (kN messages). A more effective approach decomposes random Boolean network model by regrouping lightweight agents to a "*big-agent*" and assigns a logical process (LP) to simulate each one. The *big-agent* contains a Boolean (or bit) array variable (referred as *subNetState*) to present its internal state which is constructed from all state of its lightweight agents (Figure 2). At each step, instead of sending directly state to neighborhoods, each lightweight agent updates appropriate element in this *subNetState* variable. Hence, to read state of neighborhoods, agents only need refer to the *subNetState* variables. Obviously it would be possible for neighborhoods of an agent p simulated by LP_i to be an agent q residing in another LP_j . That means agent p has to refer to q^{th} element of the variable *subNetState* of the LP_j. Thus each LP, for computing its internal state at step t + 1, must know the ones which were computed by all others LPs at step t. In other words, LPs must be synchronized. Two approaches are suggested.

The first one is based on the distributed memory model, in which each LPs after finishing its computation at step t will send its internal state to all others LPs. It continues to compute next step t + 1 only when it received *subNetState* variables from all others LPs. Suppose that the RBN model is simulated by Q different LPs, then the number of message must be sent per step is Q * (Q - 1). That reduces its performance. Detailed analyses about the performance of this approach are beyond the scope of this paper, but are discussed elsewhere.

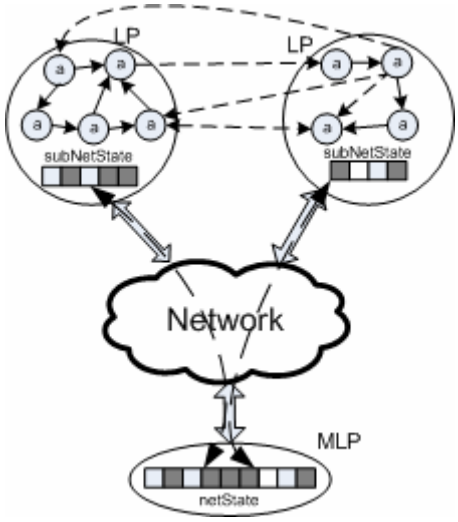

Figure 2: Decomposing a Random Boolean Network model.

The other approach is based on the share memory model, in which each LP when have finished its computation at step t, writes its *subNetState* to a *pseudo-share* memory. This *pseudo-share* memory object can be modelled by a *memory logic process* (MLP). The MLP receives all *subNetState* variables written from LPs, constructs a *netState* array that present the state of RBN and then broadcasts it again to all LPs (figure 2). In the case of Q logical processes, the number of message sent per step is 2 * Q.

By broadcasting *netState*, the MLP forces others LPs to advance together in lock step. This is a synchronous simulation and it guarantees that the causality errors does not happen but the potential for speedup is limited, especially when the RBN model is partitioned into too many sub-models, it can create a bottleneck at the memory logical process and this reduces its performance.

Let **t** be the average time value which one lightweight-agent needs for simulating a step, **δ** be the average time value for communicating the state of one agent (*it is usually a bit*) between MLP and LPs then the average simulation time T, which be computed at MLP in one step can be conjectured:

$$T_{MLP} = N\delta + \frac{N}{Q}t + \frac{N}{Q}\delta + \Delta \qquad (1)$$

Where, N is number of nodes of the modelling RBN, Q is number of logical process (so the number of lightweight agents in each LP is N/Q) and Δ is some delay coefficient of the communication, which could occur by numerous factors, for example by the waiting to process when many LPs send their internal state to

the MLP at the same time. If the ratio of **t** to **δ** is **a** then the speedup in the ideal case, Δ = 0, will be:

$$Speedup \approx \frac{Nt}{N\delta + \frac{N}{Q}t + \frac{N}{Q}\delta} = \frac{aQ}{Q + a + 1} \quad (2)$$

Clearly, the speedup depends robustly on the ratio of computation time to communication time. This ratio will be theoretic limit of speedup in our synchronous approach. This also suggests that the simulation on a multiprocessor system does not suffer from communication latency, always give a better result.

5. EXPERIMENTAL RESULTS

Currently, many languages and tool that are used for the distributed and parallel simulation are already available. Most of them use mainly the integrating of agent toolkits into a High Level Architecture (HLA) based distributed simulation and focus mainly on the interoperability between different sequential simulation toolkits not on the gaining speedups (Minson and Theodoropoulos 2004; Wang, Turner, and Wang 2005; Lees, Logan, and Theodoropoulos 2007). Some others approaches tend to obtain performance improvements but in those platforms, the communication between logical processes uses usually TCP/IP protocol which is reliable, FIFO and point- to-point (Pawlaszczyk and Timm 2006; Cicirelli, Furfaro, and Nigro 2007). Thus the broadcast from the MLP to all other LPs is essentially sequential: the MLP send continuously *netStatus* to LPs one by one. We can conjecture the speedup in this case as equation below:

$$Speedup \approx \frac{Nt}{Q * N\delta + \frac{N}{Q}t + \frac{N}{Q}\delta} = \frac{aQ}{Q^2 + a + 1} \quad (3)$$

Obviously, this *sequential-broadcast* degrades the simulation performance when model is partitioned into many logical processes.

Our experiments were performed in a network of 512MB, Intel 3.0GHZ computers. The LAN has links with 100Mb/s in bandwidth. Logical processes have been implemented with Java language, each one is executed on a separate processor. To "*write*" *subNetState* from LPs to MLP, standard RMI library was used, but we use multicast mechanism to broadcast *netState* from MLP to all others LPs. To ensure that the communication is reliable we use a timeout mechanism and a cyclic redundancy check (CRC) mechanism (Peterson and Brown 1961). The duplicate appearing will be eliminated by checking logical time (Lamport 1978) of the message received (*netState*) with local time at the receiving LP. It is clear that by using of mechanisms for checking every packet arrived, it can reduce the simulation performance but we believe that, in a Local Area Network, this reduction of performance is acceptable. Behaviours of the logical process and memory logical process are given in algorithm 1 and algorithm 2 respectively.

Algorithm 1: Logical Process

```
While true Do
   Wait netState from MLP
   If CRC value is correct Then
      If LogicalTime = LocalTime Then
         For each agent i in lightweight-agent list Do
            subNetState[i]←agent[i].step(netState)
         End for
         LocalTime++
         Send subNetState to MLP
      End if
   End if
End while
```

Algorithm 2 Memory Logical Process

```
While true Do
   Broadcast netState with its CRC value to all LPs
   Clock.Start()
   Repeat
      Wait subNetState from any LPi
      Update i^th partition of the newNetState variable
   Until received subNetState variable from all LPs
   Clock.Stop()
   netState ← newNetState
   LogicalTime++
End while
```

Figure 3 shows the fields of a packet which is broadcasted from MLP to all others LPs. The first two fields are sequence number and CRC value of the data field. The next field identifies the logical time which denotes how far in simulated time the network has simulated and the last field is broadcasted data which is network state (*netData*). Due to limitation on size of sent diagram in Java UDP, the packet size must be less than 64K, including its header, and so if network is too large, MLP will broadcast continuously *netState* variable in some different packets. In those cases, the SegNbr field identifies order of the sent packets. But in current implementation, we didn't use this field, so the largest simulated RBN contain maximum of approximate 2^{19} nodes.

Figure 3: Fields of a broadcasted packet

To estimate theoretic speedup of the simulation of large random Boolean network, we compute the ratio of average time an agent need to simulate a step to average time for communicating a single bit data between LP and MLP. We implemented a sequential simulator to simulate RBNs. 500 classical RBNs of 256K-nodes with k is 1, 4, 7 were simulated for 100 steps. The accumulated average simulation time is computed. The results of this experiment were given in figure 4.

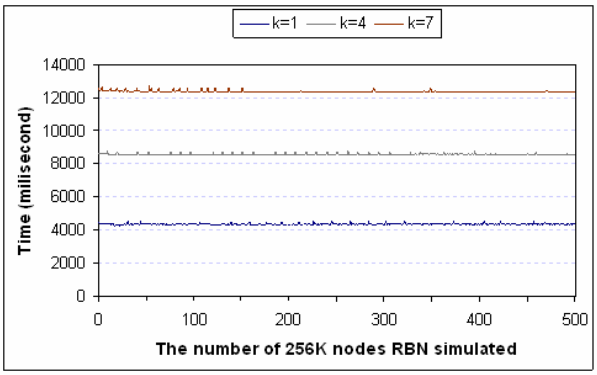

Figure 4: The average time value to simulate a 256K-nodes RBN for 100 steps

The communication time contain the time for writing data to output port at the sending process, the time for transferring data in the link and the time for reading all data from input port at the receiving process. We use two processes executing on the two separates processors: One process sends a volume of data and another process receives this data and then resends immediately it to the sender. The communication time is computed at the first process when it begins sending a volume of data until it receives whole those data. The sending process iterates this procedure in 500 times and the accumulated average time value for communicating a bit data between processes is computed. Figure 5 shows the result of this experiment in the case of volume size being 50Kb. In fact, this average time depends on the buffer size. The smaller buffer size is, the larger average time is.

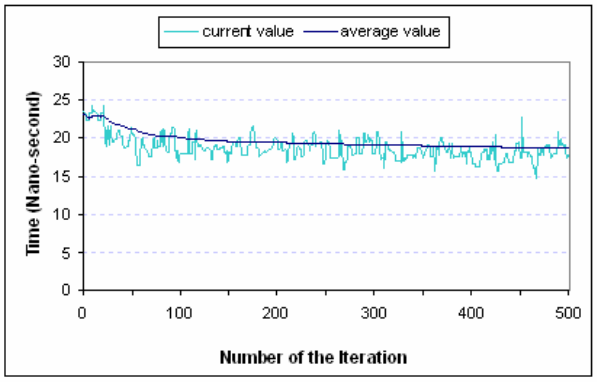

Figure 5: The average time value for broadcasting a bit of data

By using results of two experiments above, we can compute the ideal theoretic speedup according to equation 2. Figure 6 illustrates that result.

To validate these theoretic results, we simulated 500 classical random Boolean networks, each of which is composed 256K nodes. For each network, we simulate it in 500 steps. The number of logical process is equal to the number of processor. Figure 7 shows the speedup of this experiment with different values of parameter k.

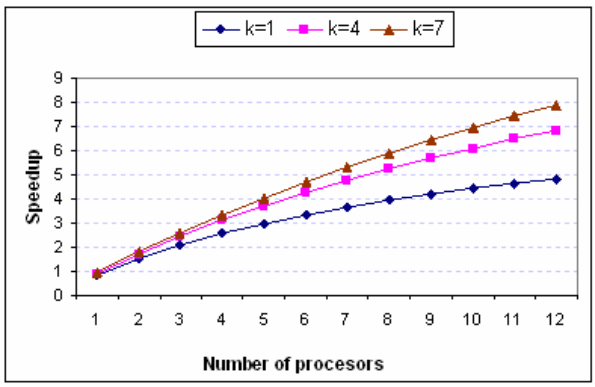

Figure 6: The theoretic ideal speedup for the synchronous simulation of large random Boolean networks

We observe that the real speedup is smaller than the theoretic one, especially when there are more processors. This is mainly due to the considerable bottleneck at the MLP. Nevertheless, in a real share memory multiprocessors system, because each LP writes really to a disjoint memory area, we believe that this bottleneck will not happen.

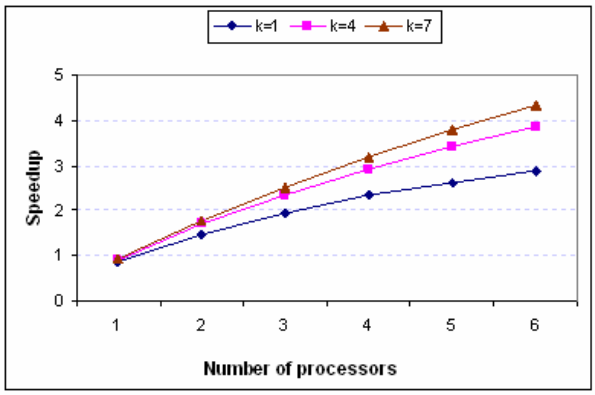

Figure 7: The real speedup for the synchronous simulation of 256K nodes RBN with different connectivities.

6. CONCLUSION

We have introduced classical random Boolean network. A simple multi-agent based approach for implementing large-scale simulation of random Boolean network has been suggested. The theoretic analysis of dependence between the performance and the ratio of computation to communication are presented. Future efforts might be directed toward simulating large random Boolean networks and other large multi agent systems on multi processors systems.

ACKNOWLEDGMENTS

I am deeply thankful to Professor Marc Bui of *the Ecole Pratique des Hautes Etudes (**EPHE**)* for his help, advice and stimulating ideas. I would like to thank the *Laboratoire de Cognition Humaine et ARTificielle (**CHArt**)* at *Ecole Pratique des Hautes Etudes*, and in particular to professor François Jouen of the *Ecole*

Pratique des Hautes Etudes for providing a proper environment for the initial preparation of this paper.

REFERENCES

Bryant, 1977. *Simulation of packet communications architecture computer systems*. Massachussets Institute of Technology.

Chandy, K.M. and Misra, J., 1979. Distributed simulation: A case study in design and verification of distributed programs. *IEEE Transactions on Software Engineering* SE-5 (5): 440-452.

Cicirelli, F., Furfaro, A. and Nigro, L., 2007. Exploiting agents for modelling and simulation of coverage control protocols in large sensor networks. The *Journal of Systems and* Software 80 (11): 1817-1832.

Derrida, B. and Flyvbjerg, H., 1987. The random map model: A disordered model with deterministic dynamics. *Journal de Physique*, 48: 971-978.

Derrida, B. and Pomeau., Y. 1986. Random networks of automata: a simple annealed approximation. *Europhys. Lett. 2*: 45-59.

Di Paolo, E.A., 2001. Rhythmic and non-rhythmic attractors in asynchronous random Boolean networks. *Biosystems* 59 (3): 85-95.

Ewald, R., Chen, D., Theodoropoulos, G., Lees, M., Logan, B., Oguara, T., and Uhrmacher, A. M., 2006. Performance Analysis of Shared Data Access Algorithms for Distributed Simulation of Multi-Agent Systems. *Proceedings of the 20th Workshop on Principles of Advanced and Distributed Simulation (PADS'06)*, 29-36.

Flyvberg, H. and N. Kjaer. 1988. Exact solution of the Kauffman model with connectivity one. Journal de Physique A 21 (7): 1695-1718.

Fujimoto, R.M, 1990. Parallel discrete event simulation. *Communication of the ACM* 33 (10): 30-53.

Fujimoto, R.M., 2000. *Parallel and Distributed Simulation Systems*. John Wiley & Sons.

Gershenson, C., 2004. Introduction to Random Boolean Networks. *Workshop and Tutorial Proceedings Ninth International Conference on Simulation and Synthesis of Living Systems (ALife IX)*, 160-173.

Harvey, I. and Bossomaier, T., 1997. Time out of joint: Attractors in asynchronous random Boolean networks." *Proceedingsof the Fourth European Conference on Artificial Life (ECAL97)*, 67-75.

Jefferson, D. R. 1985. Virtual time. *ACM Transactions on Programming Languages and Systems*, 7(3): 404-205.

Jefferson, D. R. 1990, August. Virtual time II: Storage management in distributed simulation. *Proceeding of the 9th Annual ACM Symposium on Principles of Distributed*, 75-89.

Kauffman, S., 1969. Metabolic stability and epigenesist in randomly constructed genetic net. *Journal of Theoretical Biology*, 22 (3): 437 - 467.

Kauffman, S., 1993. *The Origins of Order: Self-Organization and Selection in Evolution*. New York: Oxford University Press.

Lamport, L., 1978. Time, Clocks, and the Ordering of Events in a Distributed System. *Comm. ACM*, 21 (7): 558- 565.

Lees, M., Logan, B. and Theodoropoulos, G.K., 2007. Distributed simulation of agent-based systems with HLA. *ACM Transactions on Modelling and Computer Simulation (TOMACS)*, 17 (3): 11.

Lin, Y. and Preiss, B.R., 1991. Optimal Memory Management for time wrap parallel simulation. *ACM Transactions on Modelling and Computer Simulation* 1 (4): 283-307.

Logan, B. and Theodoropoulos, G., 2001. The distributed simulation of Multi Agent Systems. *Proceedings of the IEEE*, Volume 89, 174-186.

Macedonia, M. R., Zyda, M.J., Pratt, D.R., Brutzman, D.P. and Barham, P.T., 1995. Exploiting Reality with Multicast Groups. *IEEE Computer Graphics and Applications* 15(5), 38-45.

Minson, R. and Theodoropoulos, G., 2004. Distributing RePast Agent-Based Simulations with HLA. *Proceedings of the 2004 European Simulation Interoperability Workshop*. Paper No. 04E-SIW-046. Edinburgh

Misra, J., 1986. Distributed discrete – event simulation. *ACM Computing Survey*, 18 (1): 39-65.

Nicol, D. M., 1996. Principles of conservative parallel simulation. *Proceedings of the 1996 Winter Simulation Conference*. 128-135.

Pawlaszczyk, D. and Timm, I.J., 2006. A Hybrid Time Management Approach to Agent Based Simulation. *Proceedings of the 29th Annual German Conference on Artificial Intelligence (KI 2006)*, 374-388.

Peterson, W. and Brown, D.T. 1961. Cyclic Codes for Error Detection. *Proceedings of the IRE*, 49(1): 228-235.

Wang, F., Turner, S.J. and Wang, L., 2005. Agent Communication in Distributed Simulations. *Lecture Notes in Computer Science* 3415:11-24.

Wooldridge, M., 2002. *An Introduction to Multi Agent Systems*. John Wiley & Sons.

AUTHORS BIOGRAPHY

PHAM Dang Hai received the engineering diploma in Information Technology from the Hanoi University of Technology, Viet Nam, in 1995 and the diploma of master in Information Technology from the Institut de la Francophonie pour l'Informatique, Viet Nam, in 1997. From 1997 to 2004, he was a lecturer at the Department of computer science, Faculty of Information Technology, Hanoi University of Technology, Viet Nam. He is presently a Phd student at the Ecole Pratique de Hautes Etudes, Paris, France. His current research interest includes Parallel and Distributed simulation, Multi-agent based simulation.