A GPU-Based Simulation Kernel within Heterogeneous Collaborative Computation on Large-Scale Artificial Society

Li Zhen, Qiuxiao Gang, Guo Gang, and Chen Bin

***Abstract*—The graphic processing unit (GPU) gets strong computing ability with relatively low energy and money consumption, it has been widely used in the field of large-scale simulation and computation. Among which the CPU-GPU heterogeneous collaborative computing model has become an effective ways to solve the simulation performance of large-scale artificial society. But there are lots of problems in GPU-based ABS. The paper proposes a GPU-based conservative parallel discrete event simulation algorithm for ABS. We reorganize data structure for CPU/GPU-based heterogeneous collaborative parallel simulation, design a GPU kernel scheduling algorithm based on conservative time synchronization strategy, propose an efficient organization and scheduling algorithm for simulation event and improve execution efficiency of conservative time synchronization algorithm through the optimization of large-scale parallel time reduction algorithm. Finally, we analyze the algorithms proposed and the GPU-based simulation kernel with GameOfLife model, up to 11.2x speedup is obtained compared to CPU.**

***Index Terms*—Artificial society, ABS, CPU/GPU, heterogeneous collaborative simulation, conservative parallel.**

I. Introduction

With years of development, ABS, the Agent-Based Simulation, has become a main method of modeling and simulation [1]. In complex system area, ABS has also becomes an effective research method gradually. Social system is a typical complex system, we built the artificial society with ABS to analyze and predicts in some aspects of the real social system. In artificial society, each agent represent an independent individual of social system, it simulates and evolves in a simulated networks such as human society relationship network and environment road network. However, with the improvement of artificial society simulation, great challenges are presented to the execution efficiency of that. Take simulation for an urban society system for example, the amount of agents will be dozens of million, and there will be countless nodes and edges of all kinds of complex networks.

Parallel simulation is commonly used to improve the execution performance of simulation; it assigns simulation tasks to several physical processor units, in logical, each processor is called logical process (LP), several LPs execute simulation task collaboratively in parallel to shorten simulation time [2]. However, because of the huge investment for large scale compute cluster and the limitation of bandwidth between clusters, just increasing the cluster size and the amount of CPU cores blindly cannot obtain the corresponding performance improving to the investment. Under this background, a lot of co-processors such as GPU and MIC etc. boom in scientific computing area, because of the advantages of low power consumption, strong ability of parallel computing and relatively low prices etc. So far GPU has provided efficient solutions in fluid mechanics [3], molecular dynamics [4], bio-informatics [5] and some other areas. How to take advantage of GPU in simulation speedup also becomes hot research topic. For example, Perumalla *et al*., experiment on several typical agent-based simulation application in GPU, and obtain large speedup compared to that in CPU [6], and he also realize the simulation of diffusion system in GPU by the event driven and time step discrete event scheduling algorithm for the first time [7]. Hyungwook Park *et al*., present a framework of discrete event simulation application based on GPU [8], which divide the input event the queue into several sub-queue to enhance the degree of parallelism of event executing in GPU, it achieves good execution performance in queue simulation system [9]. Wenjie Tang, Yiping Yao *et al*., realizes a general GPU-based discrete event simulation kernel, and proposes an expansion-aided synchronous conservative time management algorithm, and a memory management algorithm, which solve the problem of memory access conflict in GPU parallel process, and raise the parallelism of simulation kernel by adding more concurrent events [10], [11]. Consequently, people try to maximize the performance of a single compute node, integrate the strong computing performance into parallel simulation. GPU, as a high performance and widely used co-processor, heterogeneous collaboratively compute with CPU is a good ways to improve the execution performance of large scale artificial society and gradually becomes a hot research topic. Aaby G and Perumalla present a "B+2R" latency hiding scheme in CPU/GPU heterogeneous computing platform to solve the bottleneck of communication between heterogeneous process units in Agent simulation [12].

Different from Wenjie Tang's work [10], [11], the paper builds a GPU-based agent parallel simulation kernel within CPU/GPU heterogeneous collaborative computation according to the characteristic of artificial society. In the second part, the paper will introduce some background

Manuscript received January 29, 2014; revised March 12, 2014. This work was supported in part by National Nature and Science Foundation of China under Grant No. 91024030 and 91324013.

The authors are with the College of Information System and Management, National University of Defense Technology, Changsha, Hunan, P. R. China, 410073 (e-mail: listyle1991@gmail.com)

DOI: 10.7763/IJMO.2014.V4.374

knowledge related to large-scale artificial society, GPU and CUDA; in the third part, the discussion is center on the basic data structure of GPU-based parallel agent simulation kernel, kernel scheduling algorithm, event management and scheduling algorithm and time reduction algorithm, etc. In the fourth part of the paper, it takes a specific GameOfLife experiment case to test and verify the performance of the GPU-based simulation kernel; lastly, the fifth part concludes the whole paper and the discussion of the research work in future.

II. Background

A. Agent-Based Artificial Society

The origin of artificial society research is the social-cybernetics presented by the founder of cybernetics Norbert Wiener, by adopting the idea of bottom-up, it builds complex society system model using local connection rules with technology of agent-based simulation, and becomes a new method of social science research. With this method, the computer should abstract each individual into individual model called agent. Then plenty of agents interact by rules, which will bring emergent of some macro behaviors, thus researching on the emergent can help understanding and explaining the macro phenomenon.

Generally, we abstract the process above as sense-think-act paradigm, in every simulation time step, agent senses the information of environment (including information of environment model and the status info of other connected agents), then it makes estimation and decision with these information and its action rules, finally it acts to change status of itself, environment and even other entities.

It can be seen from the above paradigm that agent is connected with network topologies which specifies the action ranges and form of agent's sense and action. Generally there are 5 kinds of topology structures between Agents: "Soup" Model, Cellular Automata, Euclidean 2D/3D Space, Network topology, Geographic Information System [13].

Most social system researched with Agent-based artificial society simulation are large-scaled, Take the GSAM program [14], [15] which armed at disease transmission research as an example, a disease transmission model on a global scale is built, it also runs successfully based on agent-based simulation. So, compared to other normal agent-based simulation, agent-based artificial society has its own characteristic:

1) The number of agent is enormous, and the network topology is huge and complex.
2) The action rules of agent have the characteristics of high unity and concurrency. A large number of agents are used to simulate the same type of people in artificial society, these Agents step in a unified global time and process the states evolution with the same kind of action rules.

B. GPU and CUDA

GPU (Graphic Processing Unit, GPU) was firstly appeared as a graphic processing unit, it was used to graphics rendering, with the development of hardware and software technology, people come to realize that GPU has its own advantages and characteristics in scientific computing area over CPU. In recent years, the computing power of GPU has been growing at more than the Moore's law. Tianhe-I which has been the first of supercomputers gains great computing performance with thousands of GPU working cooperatively.

Take the GPU made by NVIDIA co. for example [16], on the hardware structure, there are several SM (SM, Streaming Multi-processors) in a GPU process unit, and a SM contains several SP (SP, Streaming Processors), which is commonly called the core of GPU. These cores execute currently in SM and share the same global memory, and each SM contains a shared memory so that the SP of the same SM can share it. What's more, there is global constant memory and texture memory to be shared.

On the programing structure, the NVIDIA co. design a specialized development library for GPU called CUDA, which can offer users friendly program interface to hide the details of the underlying hardware. CUDA is a C-like language and easy to learn. Although it can only be used in GPU of NVIDIA co., it has still been widely used

CUDA can be divided into three layers based on hardware structure: grid, block and thread. Take, for example, GPU with the computing performance of 2.0, a grid is commonly composed of 65535*65535 blocks, and a block is composed of 1024 threads. Thus, CUDA can map blocks in grid to different SMs and map threads in blocks to different SP. Fig. 1 shows the general programming model of CUDA.

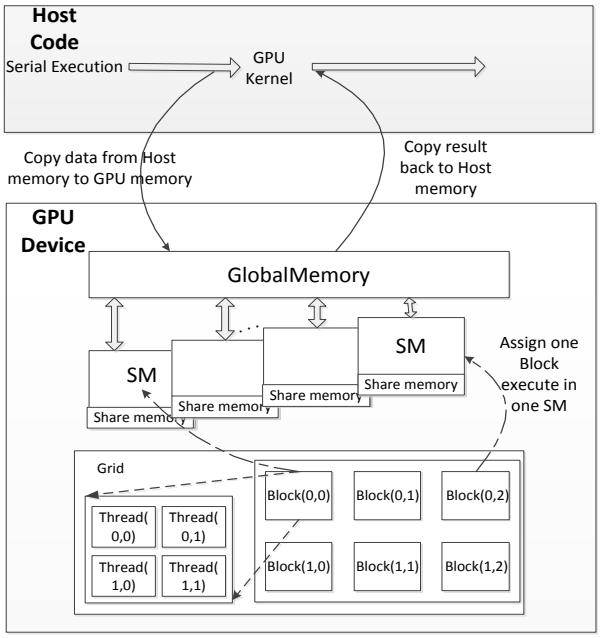

Fig. 1. CUDA programming model.

III. GPU-Based Simulation Kernel

The characteristic of artificial society bring severe challenges to the execution of simulation. Integrating GPU into simulation computational resources, in which GPU is treated as an isolated computation node to compute models collaboratively with CPU, is a good way to solve the execution efficiency problem of large-scale artificial society. The paper designs a GPU-based conservational parallel discrete event simulation kernel and realizes it.

A. Data Structure and Kernel Scheduling Algorithm

The data structure supports discrete event simulation; it includes event and organization of event queues. But furthermore, in order to support CPU/GPU-based heterogeneous collaborative computation, we need to redesign the data structure as Fig. 2 shows.

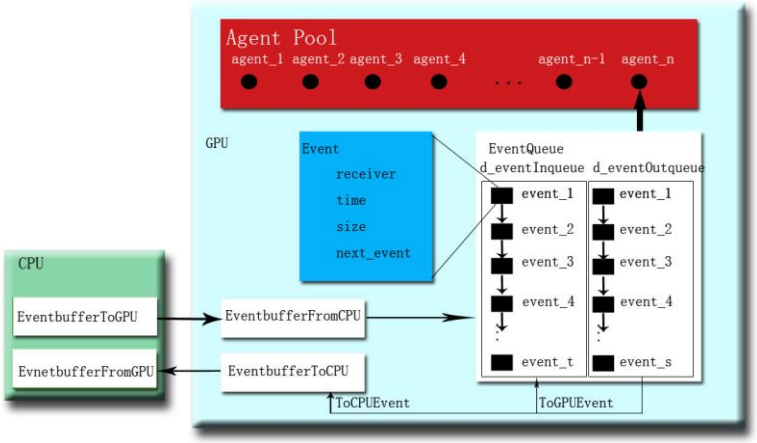

Fig. 2. The organization and mangerment of agents and events in GPU memory.

Agents are organized into an agent pool in GPU, each agent maintains an input event queue denoted as d_Event In Queue and an output event queue denoted as d_Event Out Queue, both event queues are arranged in time order. The former one is used to receive all the events which are sent to its corresponding agent, the latter one is used to caches the events produced in running process for its corresponding agent. The organizational architecture is designed in the consideration of two aspects reason: the first, to separate the input event queue from the output event queue can improve the parallel execution efficiency of GPU while don't considerate the interactivity with other threads; the second, each agent maintaining its own event queue bypasses memory access conflict of parallel event and make for promoting the degree of parallelism. In the whole process of simulation, event flow includes CPU to CPU, CPU to GPU, GPU to GPU, and GPU to CPU. Therefore in GPU-based simulation kernel, event scheduling involves event interaction between CPU and GPU. Accordingly we design two event buffers in each device to realize event interaction between CPU and GPU, which are Event Buffer From CPU and Event Buffer To CPU in GPU and Event Buffer To GPU and Event Buffer From GPU in CPU.

GPU-based simulation kernel is scheduled by CPU; Fig. 3 illustrates the GPU-based simulation kernel scheduling frame. In the Initialization phase, CPU is responsible for initialization of GPU device and creation of the data structure in GPU-based simulation kernel. In the Pre Initialize Model phase, the GPU is launched to execute model's initialization function, which initializes the agent objects and event queues. After completing the initialization works, all the event queues contain an initial event whose simulation time is zero. Then in the Reduce Time phase, it take the first event out from each event queue, and launch a GPU kernel to reduce their simulation times, the reduction result is the minimal simulation time among the events, according to conservative time synchronization algorithm, the minimal time is the global simulation time. Then judge whether the global simulation time is larger than simulation stop time, if so, jump to the END phase; if not, turn to Execute Model phase. In Execute Model phase, GPU kernel is launched to executed agent model, each thread execute in parallel to see if the event time of its corresponding agent's event queue is less than the global simulation time, if so, the thread preform an agent update. After all the thread finish its work, the Synchronize Event phase begin, in order to avoid multithreads currently access the same event queue in the running process, the events which is new generated should be synchronously received. After Synchronize Event, turn back to Reduce Time, and repeat the operation above.

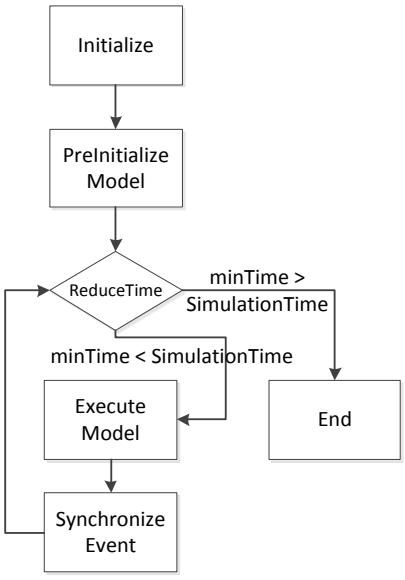

Fig. 3. GPU-based simulation kernel scheduling flow.

B. Event Organization and scheduling

In CPU/GPU-based heterogeneous collaborative parallel simulation engine, each compute node (GPU or CPU) need time synchronization and event communication. So, the event organization and scheduling in GPU should involve two aspects: the first is event communication between CPU and GPU, another one is event communication in GPU.

1) The event communication between GPU and CPU is managed by Event Buffer From CPU and Event Buffer To CPU. When the agents perform update operation, they just take the first event out from their own input event queue d_Event In Queue, so when CPU send event

to GPU, firstly it put all the events whose receiver is in GPU into Event Buffer To GPU, then perform a data transformation, copy the data from Event Buffer To GPU to Event Buffer From CPU. After that the agent objects in GPU read the events in Event Buffer From CPU to see if there are events whose receiver is itself in parallel, if so, push the events into its d_Event In Queue.

2) The event communication between GPU is performed by the agents whose event receiver is also in GPU. The agents in GPU take the first event out from d_Event In Queue to perform update operation. In update process, the events which are new generated are directly pushed into d_Event Out Queue. In Synchronize Event phase, the events whose receiver is in GPU are sent from the d_Event Out Queue to the corresponding d_Event In Queue. But, in large-scale agent simulation, many agents may send event messages to the same agent unavoidable. Then the problem is to solve memory access conflict when multithreads write to the same d_Event In Queue in GPU. The paper gives an event message output algorithm to solve the problem above. We make use of atomic add operation in CUDA, that operation can be regarded as a synchronization point at which parallel threads perform in turn. Thus, every d_Event In Queue contain a public variable denoted as queue Wait and a public unsigned integer array denoted as temp Event, when multithreads access d_Event In Queue currently, the atomic add operation on queue Wait can record the order of each thread which is accessing the queue. Then each thread get a numeral ID which ranks in order and assign address of event performed by itself to the ID_{th} variable of temp Event. After that, every agent performs its d_Event In Queue; push the events in temp Event into d_Event In Queue in parallel. Fig. 4 illustrates the pseudo code of the event output algorithm.

```
For all thread t in parallel do
   int wait=atomic Add(&queue Wait_t,1);//return the value of queue Wait when it comes
   temp Event_wait = (unsigned int)e;
end for all
for all thread t in parallel do
// each thread handle their event address array and push it into queue serially
   Shared__ int wait Number;
   Wait Number = queue Wait_t;//get the whole thread number in queue
   For i:=0 to wait Number
      Event *e;
      e = (Event *)temp Event_i;
         Push e to d_Event In Queue_t
      Queue Wait_t= 0;
   End for
End for each
```

Fig. 4. GPU-based event output algorithm.

C. Large-Scale Parallel Time Reduction Algorithm

The core matter of GPU-based parallel discrete event simulation is to insure correct causality between events. In conservational time synchronization algorithm, that is to give correct global simulation time. So each step of simulation performs an operation of global event time reduction. When dealing with large-scale artificial society, the scale of event scheduling can easily reach sever million. So, to decrease the execution time of reduction is the crux of improving the performance of GPU-based simulation kernel.

Reduction algorithm is a typical recursive algorithm, so combining the programming feature of GPU, the large-scale parallel time reduction algorithm in GPU should take two problems into account: the first one is how to realize recursion in GPU; the second is how to realize the global synchronization of all the threads. Fig. 5 illustrates the pseudo code of the time reduction algorithm in GPU.

```
   Forall thread t in parallel do//T stands for the times to be reduced sortNUM stands for the number to
be sorted
      int cycle;
      while sortNUM/cycle > 1 do
         for i:=t to sortNUM/(cycle*2)
            if 2*i*cycle < simulationNum && (2*i+1)*cycle < simulationNum
               if T_(2*i*cycle) > T_((2*i+1)*cycle)
                  T_(2*i*cycle) = T_((2*i+1)*cycle);
               endif
            endif
         endfor
      __syncthreads();
      cycle = cycle*2;
      endwhile
      minTime = T_0;
   endforall
```

Fig. 5. GPU-based large-scale time reduction algorithm.

As all we know, recursive algorithm is composed of many reduction steps. The number of times to be reduced gives; the number of steps knows. But the hardware structure of GPU determines that GPU could not support recursion well. In

order to realize it, a bad way is to let CPU launch GPU kernel recursively, but it increase the burden of CPU, and decrease parallel execution efficiency of GPU by decompose reduction into server GPU kernel function, thus at the same time increase the time for kernel launch. Based on these considerations, the paper set a control variable denoted as cycle, when the algorithm begins, cycle is initialized by 1, after each reduction, multiply cycle by 2, and perform the reduction step in circle until cycle is larger than the number of reduction steps.

Because of the order being launched and speed of execution, the time used in one reduction step of work threads in GPU is not the same. So after each parallel reduction step, a global synchronization of all the threads is needed. While the synchronization operation is limited in one SM, namely only the threads in one block can be synchronized in running process. As the analysis above, we design a reduction algorithm that all the parallel operation is finished in one block. Suppose the thread number in one block is denoted as THREADNUM, then each work thread goes through the time array by increment THREADNUM. In theory, the reduction algorithm can be any scale.

IV. Experiment

In order to verify the performance of GPU-based simulation kernel, the paper design a Game of Life model to be scheduled and processed in GPU. In Game of Life model, all the agents are organized into a 2-dimension grid, whether the agent in the grid is alive is determined by the number of agents alive in its 8 neighborhoods. In the process of evolvement, an agent relives when there are 3 agents alive in its 8 neighborhoods, and dies when the number of living agents larger than 3 and smaller than 2.

The experiment performs on a 2-dimension 100*100 grid, altogether 10000 agents. Simulation runs 120 seconds. Experiment hardware condition is Intel Xeon E5645 2.4GHZ 12 cores CPU and a NVIDIA Tesla C2050 GPU, system memory is 64GB. Tesla C2050 has 448 stream processers, compute ability is 2.0, and global memory is 2GB. As the software condition, MPICH2 is used to realize communication between CPU cores, GPU uses driver and SDK of CUDA 5.0 version.

The Game of Life model is experimented on 1core, 2 cores, 4 cores, 8 cores and GPU separately; experiment result is showed on Fig. 6. Different color blocks on the figure represent different important phase of the whole simulation process, they are initialization phase, model execution phase, time reduction phase, and event receive synchronization phase. From a general view, the whole simulation time of GPU obtain a speedup of 11.2x compared to serial execution of 1 core CPU, 2.9x speedup compared to 2 core CPU's parallel execution, and equal to the performance of 4 cores CPU. The crosswise comparison on CPU and GPU shows that in CPU the time cost on event receive synchronization phase take up nearly 90 percent of the whole time, but in GPU nearly 100 percent of the whole time is used in model execution phase.

Combing the characteristic of hardware and discrete event simulation algorithm, we can analyze and explain the result. For GPU, there are hundreds of cores in physical and thousands of threads working in parallel logistically. In our GPU-based discrete event simulation algorithm, each agent maintains its own event queue, thus in the simulation process, event queues are assign on a large number of logical threads, so in each event queue, the number of event is relative small, it only needs a parallel execution cycle to complete the event receive synchronization phase. But to the contrary, the event queues in discrete event simulation on CPU are assigned on logical process, but due to the number of cores in CPU is relatively small, when the simulation scale is large, the event queue on each logical process shall contain many events to be process in event receive synchronization phase, Thus each logical process should deal with lots of events serially. When the model execution operation is relatively easy, the time cost in the event receiving synchronization phase will then take up most time of the whole simulation as Fig. 6 shows.

The NVIDIA GPU architecture is built on Streaming Multiprocessors (SMs), and SMs use an architecture called SIMT to organize work threads. The SIMT works in parallel programs well, but it fails in control flow instruction which needs lots of branch prediction. At the same time the performance of single thread in GPU is much less than that of a single CPU core. It can obtain a relatively higher speedup compared to CPU when large scale threads (millions) are launched. Because when lots of threads are launched, the latency of single thread execution can be eliminated by switching between threads. In the our experiment, the scale of simulation is just 10 thousand which is relatively small, so experiment result presents the model execution time in GPU take the most time of simulation, and it is even larger than that in 1 core CPU.

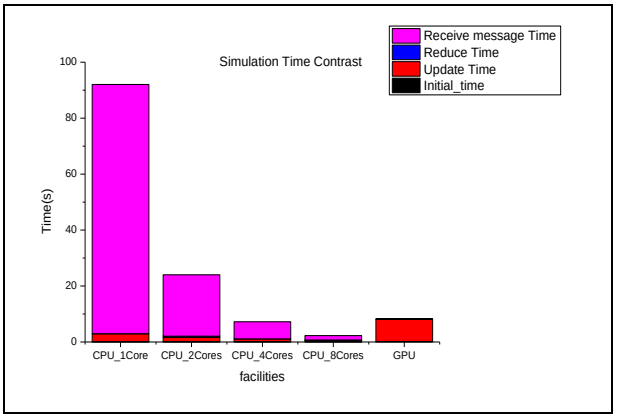

Fig. 6. Simulation time contrast on different operating environment.

V. Conclusion

Facing on CPU/GPU-based heterogeneous collaborative large-scale agent simulation, the paper analyzes and illustrates the design and realization of GPU-based parallel discrete event simulation kernel from the organization of data structure, kernel scheduling algorithm, event organization and scheduling algorithm, and time reduction algorithm. Experimenting on 10 thousand agents GameOfLife model to test performance of GPU-based simulation kernel, and up to 11.2x speedup is obtained compared to that on CPU. The experiment result shows that GPU has its own advantage and

feature on parallel discrete event simulation. In the future work, we will conduct a study on aspects blow.

Design efficient Communication and synchronization model between CPU and GPU. In heterogeneous collaborative simulation, the low bandwidth between different devices is the bottleneck of simulation when evolving communication between CPU and GPU, so an efficient ways to realize communication between CPU and GPU is a key point to speed up heterogeneous collaborative simulation.

Improve GPU-based simulation kernel execution efficiency. Different algorithm designing may lead different execution efficiency especially on GPU. How to adapt the SIMT programming architecture is the key point of improving algorithm efficiency.

REFERENCES

[1] M. J. North and C. M. Macal, *Managing Business Complexity: Discovering Strategic Solutions with Agent-Based Modeling and Simulation*, Oxford University Press, 2007.

[2] R .M. Fujimoto, *Parallel and Distribution Simulation Systems*, New York: Wiley-Interscience, 2000.

[3] J. Tölke, "Implementation of a lattice boltzmann kernel using the compute unified device architecture developed by NVIDIA," *Computing and Visualization in Science*, vol. 13, pp. 28-39, 2010.

[4] T. Narumi, R. Sakamaki, S. Kameoka, and K. Yasuoka, "Overheads in accelerating molecular dynamics simulations with GPUs," in *Proc. Parallel and Distributed Computing, Applications and Technologies, PDCAT 2008. Ninth International Conference*, 2008, pp. 143-150.

[5] Y. H Li and W. H. Zhu, "GPU-accelerated multi-scoring functions protein loop structure sampling," in *Proc. Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium*, 2010, pp. 1-8.

[6] K. S. Perumalla and B. Aaby, "Data parallel execution challenges and runtime performance of agent simulations on GPUs," in *Proc. Agent-Directed Simulation Symposium, 2008 spring simulation multi conference*, 2008, pp. 116-123.

[7] K. S. Perumalla, "Discrete-event execution alternatives on general Purpose graphical processing units (GPGPUs)," in *Proc. The 20th Workshop on Principles of Advanced and Distributed Simulation*, 2006, pp. 74-81.

[8] H. Park and P. A. Fishwick, "A GPU-Based application framework supporting fast discrete-event simulation," *Simul-T Soc Mod Sim*, vol. 86, no. 10, pp. 613-628, 2010.

[9] H. Park and P. A. Fishwick, "An analysis of queuing network simulation using GPU-Based hardware acceleration," *ACM Transactions on Modeling and Computer Simulation*, vol. 21, no. 3, pp. 18-39, 2011.

[10] W. J. Tang and Y. Yao, "A GPU-based discrete event simulation kernel." *Simulation: Transactions of the Society for Modeling and Simulation International*, vol. 89, no. 11, pp. 1335-1354, 2013.

[11] Tang Wenjie and Y. P. Yao, "An Expansion-aided synchronous conservative time management algorithm on GPU," in *Proc. the 2013 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, 2013, pp.367-372.

[12] B. G. Aaby and K. S. Perumalla, "Efficient simulation of agent-based models on multi-GPU and multi-core clusters," in *Proc. the 3rd International ICST Conference on Simulation Tools and Techniques (SIMUTools'10)*, 2010, pp. 15-19.

[13] C. M. Macal and M. J. North, "Agent-based modeling and simulation," in *Proc. Winter Simulation Conference*, 2009, pp. 86-98.

[14] J. Parker and J. Epstein, "Distributed platform for global-scale agent-based models of disease transmission," *ACM Trans. Model. Comput. S.*, vol. 22, no. 1, pp. 20-45, 2011.

[15] C. L. Barrett *et al*, "EpiSimdemics: an efficient algorithm for simulation the spread of infectious disease over large realistic social networks," in *Proc. the 20th ACM/IEEE Conf. on Supercomputing*, 2008, pp. 282-290.

[16] *Cuda C Programming Guide*, 5th ed., NVIDIA Co., 2012.

Li Zhen was born in Jiangxi China, 1991. He is currently a post graduate student at College of Information System and Management, National University of defense technology, China. He received his B.Sc. degree in College of mechatronic engineering and automation, National University of Defense Technology. His research interests include agent-based modeling and simulation, high performances simulation.

Qiu Xiao Gang is a professor at Institute of Simulation Engineering, College of Information System and Management, National University of Defense Technology, China. His research interests include agent-based modeling and simulation, parallel and distributed systems, emergency management, and knowledge engineering.

Guo Gang is an associate professor at Institute of Simulation Engineering, College of Information System and Management, National University of Defense Technology, China. His research interests include parallel discrete event simulation, high performance simulation.

Chen Bin is a lecturer of National University of Defense Technology, Changsha, China. His current research interests include modeling and simulation about complex system.