Proceedings of the 2013 Winter Simulation Conference
R. Pasupathy, S.-H. Kim, A. Tolk, R. Hill, and M. E. Kuhl, eds

A HOLISTIC ARCHITECTURE FOR SUPER REAL-TIME MULTIAGENT SIMULATION PLATFORMS

Toyotaro Suzumura

IBM Research – Tokyo
Tokyo Institute of Technology
Tokyo, JAPAN

Hiroki Kanezashi

Graduate School of Information Science and Engineering, Tokyo Institute of Technology
Tokyo, JAPAN

ABSTRACT

This paper describes a holistic architecture for super real-time multi-agent simulation platforms by implementing a complete and integrated simulation stack including a simulation runtime and an application layer that can be used for such situations as traffic simulations. With our prototype system, the first experiment tested the performance scalability when simulating millions of agents on 1,536 CPU cores over 256 nodes. By compiling the X10-based agent simulation system into C++ using MPI, we could run 600 simulation steps in only 78 seconds, which is nearly 10 times faster than real time. We then tested a national network spanning Japan, which was able to simulate a 100 million agents and at near-real time while using 128 nodes. This was the first attempt to deal with such a large number of agents and shows that this infrastructure could be used for large-scale agent simulations in various fields.

1 INTRODUCTION

Research in multi-agent simulation with simulating millions of agents is contributing to human society and the global economy in such areas as environmental protection, the analysis of human brain characteristics, and our daily lives (Paulocci 2012). Previous research has described how distributed agent simulation platforms can be applied to such goals, but related projects have shown their method scalable to the behavior patterns of millions of agents (Sanjay 2005; Parker 2005; Theodoropoulos 2006; Sislak 2009; Kalyan 2011) but we need to pursue more high performance scalability. We are working on the first infrastructure platform that will be able to simulate millions of interacting agents. Our previous research introduced an X10-based agent simulation platform with tens of millions of agents suitable for traffic simulations. The performance was not further scalable due to various problems involving workload imbalances and global synchronizations. In this paper we present a new simulation stack with a runtime and the application layer for X10 (Saraswat 2007) a state-of-the-art PGAS language. We tested our platform on highly distributed systems, approaching millions of agents running at near real time. We are reporting on two major experiments. The first set of results showed performance scalability up to one million agents running on 256 nodes with 1,536 CPU cores. By compiling an X10-based agent simulation system into C++ and MPI, we could run 600 simulation steps in only 78 seconds, which is nearly 10 times shorter than real time. Our second experiment used a national network spanning Japan as the underlying infrastructure for the agents, and our near-real-time simulation of hundreds of millions of agents required only 128 nodes. This is the first published attempt to deal with such large numbers of agents and we believe that this infrastructure will be used for large-scale agent simulations in various fields.

Here is a summary of the remainder of this paper. First, an overview of the X10 language as the underlying language of XAXIS is presented in Section II. Section III describes the design and implementation of the XAXIS system. Section IV describes our performance evaluations and scalability. Our conclusions and future directions are addressed in Section V.

2 X10 – HIGHLY PRODUCTIVE PARALLEL AND DISTRIBUTED LANGUAGE

Since we use X10 as the underlying programming language of our simulation platform, XAXIS, we provide an overview of the X10 language in this section.

X10 is a novel parallel distributed programming language that IBM Research is developing as an open source project. It allows the development of highly efficient applications with high throughput in a distributed system in mixed environments with various configurations of multiple cores, accelerators, and other resources. In environments with many execution cores, it is important to assess the parallelism and memory configurations. X10 offers a global address space that is partitioned into multiple places, called the Partitioned Global Address Space (PGAS).

A place is an abstraction for the locality of memory, and it typically corresponds to one compute node with two or more CPU cores. An activity can be generated dynamically in each place as an asynchronous execution subject equivalent to a lightweight thread. An activity can be generated using the syntax of an "*async"* sentence, and can be moved to other places using an "*at"* sentence. Conventionally, to run a simulation using a massive computer where each node has multiple cores and that are combined using a network, the Message Passing Interface (MPI) is used as the messaging mechanism among nodes. Programming models such as OpenMP and POSIX threads are used for thread-parallel execution in one node. However, the concepts of "*Place*" and "*Activity*" in X10 support executive operating systems on a massive computer cluster using a unified programming model with high throughput.

X10 supports parallelism for both concurrent and distributed code. Parallel code uses asynchronous un-named activities created by the *async* struct and waits for activities to finish by using the "*finish*" construct. Distributed code is run in multiple places that do not share memory, and therefore objects are deeply copied from one place to another when captured by the "at" struct. Copying is done by first serializing the object into a buffer, sending the buffer to the other place, and then de-serializing the buffer at the other place.

Currently the X10 compiler is implemented as a translator for other programming languages. X10 can be compiled to Java or C++ environments, as shown in Figure 1. An X10 environment running on Java is called Managed X10, while X10 on C++ is called Native X10. X10 compiler has two parts: a front-end and a backend. The front-end analyzes the X10 source and checks the types to create an AST (Abstract Syntax Tree). The backend generates code from the AST as Java code or C++ code.

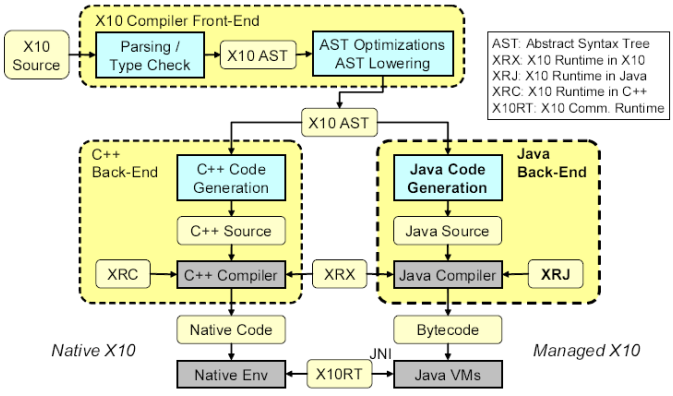

Figure 1: X10 Runtime Overview - Native X10 and Managed X10

3 XAXIS: SCALABLE X10-BASED AGENT SIMULATION MIDDLEWARE

XAXIS (X10-based Agent eXecutive Infrastructure for Simulation) (Suzumura 2012) is a highly scalable multi-agent simulation middleware that allows application developers to run their applications productively on large-scale environments. The programming model of XAXIS was derived from the Java-based agent simulation middleware called IBM Zonal Agent-based Simulation Environment (ZASE) (Yamamo-

to 2007). This section reviews overviews of the agent model, its software stack, and the implementation model of XAXIS.

3.1 XAXIS Agent Model

In XAXIS each agent has internal states, handles messages, and updates its own states as needed. Messages are sent by a simulator or by other agents. When a message object is delivered to an agent, a callback method of the agent is called. A returned message object will be sent back to the sender. The XAXIS framework provides functions to create and to delete agents. Developers can add services to an agent runtime. Agents can search for the reference to a service and can directly invoke methods of services. Each service object also has functions similar to those of agents. Messages are exchanged among agents, simulation runtimes, and agent runtimes. XAXIS provides point-to-point messaging and multicast messaging. XAXIS also provides message distribution and aggregation functions. When a simulator sends a multicast message to an agent runtime, the agent runtime will distribute the message to agents. An agent runtime aggregates the reply messages from agents into an aggregated message and sends that message to a simulator.

3.2 XAXIS Software Stack

The full software stack of XAXIS is illustrated in Figure 2. An advantage of adopting X10 is that the XAXIS APIs can be implemented in a highly productive manner without implementing thread management or a messaging layer for the distributed environments, since the X10 language itself includes those functions at the language level. The software stack in Figure 2 includes two bridges, Java bridge and X10 bridge depending on whether Java or X10 is being used. The Java-based bridge is for Java-based simulations on XAXIS. This is because Java is still more of a commodity language than X10 when developers implement simulations. However, we also provide an X10-based bridge so that the full software stack can be implemented entirely in X10 and compiled into an X10 C++ backend with high performance communication libraries such as MPI for developers who want to implement their simulations directly in X10.

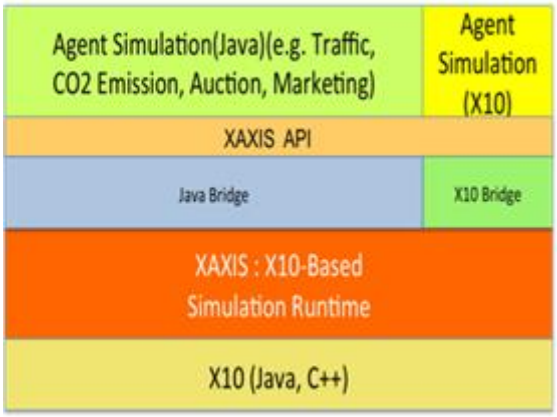

Figure 2: XAXIS Software Stack

3.3 XAXIS Implementation with X10

Figure 3 shows how the simulation model of XAXIS on a distributed environment is implemented with X10. An agent manager manages the agents running asynchronously in X10 at each place in an agent simulation. For communication with the agents at other places, the place managed by a particular agent manager is selected based on the agent's identifier, and a communication message is sent there. This communication message is hidden from the users, so the simulation developers can concentrate only on the logic of the agents. A message that is developed using X10 can be transmitted and received in XAXIS without explicit message communication in the distributed computer, but simply by calling the method of

the agent managers at other places using the *at* syntax. A mechanism for communication between places using activities that implement asynchronous threads and *at* syntax allows us to easily mount the simulation execution environment.

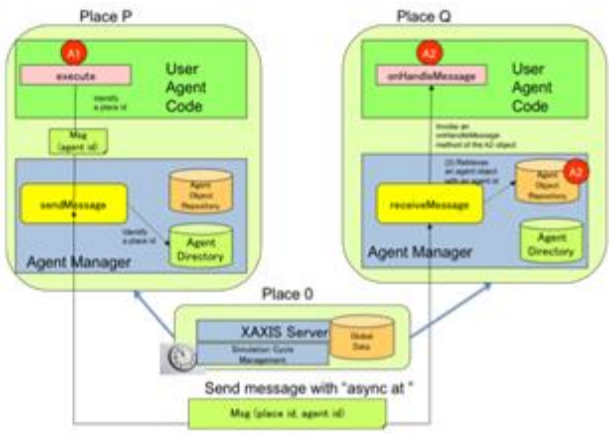

Figure 3: Simulation Model Implementation with X10

4 A NATIVE X10 APPROACH TO OPTIMIZING PERFORMANCE AND SCLABILITY OF DISTRIBUTED AGENT SIMULATIONS

To run much larger agent simulations or make it more high performance, we had to drastically improve the existing simulation software. First, we rewrote all of the programs in X10, not just the simulation platform XAXIS, but also the traffic application, IBM Mega Traffic Simulator (namely Megaffic) (Osogami 2013). By rewriting for native executables by using C++ source code, the need for a Java VM. Second, we reduced communication volume to gain scalability and high performance when running many nodes. We also improved procedures related to migration of vehicles. The main reason why the simulation time is not reduced with more computation nodes is communication overhead between nodes. We tried to reduce communication volume by adjusting the frequency and size of each migration.

4.1 The Develeopment of Megaffic and XAXIS

We are developing traffic simulation programs using this agent-based model. We began by porting Megaffic to the XAXIS simulation platform Because Megaffic was implemented in another simulation platform written in Java, we had to compile XAXIS as a Java program to be compatible with Megaffic. First we ran simulations with relatively small road networks like a city in Japan and a relatively small number of vehicles. However, even for small scale traffic agent simulation, the scaling effects were not efficient. The main constraint is the connection overhead for synchronization with other nodes and for sending agent data. In particular with Java, the data transmission procedures are too slow because the Java objects have to be serialized and deserialized when being sent to other nodes.

4.2 Implementing Fully X10-Based Traffic Simulation

Generally speaking, a native executable program is faster than a Java program because Java runs in a virtual machine although dynamically compiled with Java Just-in-time compiler. In contrast, Java is more productive for programmers than C++. The same X10 source code can be compiled into Java or C++, depending on the environment and user applications. However, because our Megaffic program is written in Java, it must be compiled into Java with Managed X10. Java programs run any machines with a Java virtual machine, but many specialized supercomputers do not support the Java virtual machine. Even when Java is supported on a supercomputer, it rare pushes the machine to its full cap.

We decided to rewrite the Java source code in XAXIS and Megaffic into X10 source code. Fortunately, because X10 was designed to combine high performance with high productivity, the syntax of X10 is similar to Java, is the porting work is not too difficult. Then we compiled the traffic simulation to a native executable from the C++ source code.

4.3 Reducing Communication Among Nodes

In the previous simulation XAXIS platform and Megaffic application (Suzumura 2012), the simulation did not scale well even with fewer than 100 places. This was because the increasing numbers of vehicles that were migrating to other nodes divided the road network into more partitions, and there was too much associated copying of large objects.

We could not reduce the number of migrations, so we simplified the migrating objects with vehicle data. In X10, if a migrating object has non-primitive field variables, then the X10 runtime will copy all of the objects referenced by that object. The vehicle agent object has referring types of cross points, roads, and other model classes, so even if one vehicle copies to a neighboring node, unnecessary objects will be copied and large amounts of the heap memory and connection bandwidth will be wasted. Therefore, we changed the data types of the vehicle data that has to be copied. All of vehicle data is converted to Long or Double primitive values. Example types such as cross points and road objects only require the ID information, so the ID data is extracted and sent from such objects.

5 PERFORMANCE EVALUATION

In this section we describe a performance evaluation of the pure X10-based simulation system on supercomputers with a national network and millions of agents and describe the scalability.

5.1 Evaluation Environment - TSUBAME 2.0 Supercomputer

A Pure X10-based traffic simulation was run on XAXIS using the TSUBAME 2.0 supercomputer at the Global Scientific Information and Computing Center of the Tokyo Institute of Technology. TSUBAME 2.0 is a production supercomputer operated by the Global Scientific Information and Computing Center (GSIC). TSUBAME 2.0 has more than 1,400 compute nodes interconnected by high-bandwidth full-bisection-wide Infiniband fat nodes, which differ in their local memory capacity. All of the compute nodes share a scalable storage system with a capacity of 7 PB.

Each TSUBAME 2.0 node has two Intel Westmere EP 2.93-GHz processors (Xeon X5670, 256-KB L2 cache, 12-MB L3), and 50 GB of local memory. The operating system is SUSE Linux Enterprise 11. Each node has a theoretical peak of 1.7 teraflops (TFLOPS). The main system consists of 1,408 computing nodes, and the total peak performance can reach 2.4 PFLOPS. Each of the CPUs in TSUBAME 2.0 has six physical cores and supports up to 12 hardware threads with Intel's hyper-threading technology, with a theoretical peak of 76 gigaflops (GFLOPS).

The interconnect that links the 1,400 computing nodes with storage is the latest QDR Infiniband (IB) network, which has 40 Gbps of bandwidth per link. Each computing node is connected to two IB links, so the communication bandwidth for the node is about 80 times larger than a fast LAN (1 Gbps). Not only the link speed at the end-point nodes, but the network topology of the entire system heavily affects the performance for large computations. TSUBAME 2.0 uses a full-bisection fat-tree topology, which accommodates applications that need more bandwidth than provided by such topologies as a torus or mesh. One of the key features of the TSUBAME 2.0 architecture is high-bandwidth hardware that transmits data efficiently. To sustain high rates of intra-node communications, the memory bandwidth is 32 GB/s for each CPU.

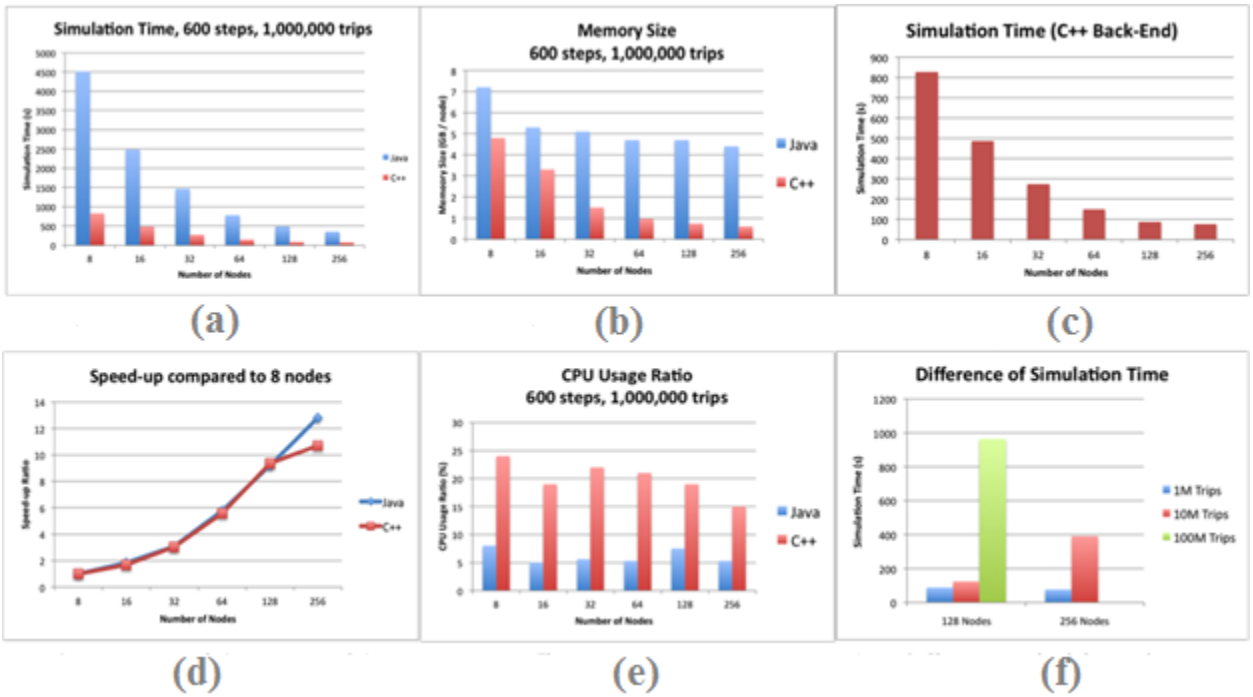

Figure 4 : Performance data (a) Simulation Time for 1 Million Trips, (b) Memory used per Node, (c) Simulation Time for 1 Million Trips with C++ Backend, (d) Speed-up Ratio compared to 8 Nodes, (e) CPU Usage Ratio, (f) Simulation Time in Relation to Nodes and Trips

5.2 Experimental Road Network Data and Graph Partitioning for Multi-Node Environment

We used the national road network data for Japan from the Open Street Map. It contains almost one million cross points and over two million roads. It is impossible to store all of the road network elements at each node, so we divided the road network by the number of nodes used in each experiment. Partitioning the whole network in chunks provides another positive effect in that the time for parsing the road network data on each compute node is greatly reduced.

When dividing the road network graph, we used METIS (Karypis 1998), which is a graph partitioning library to minimize the edge-cut value. Each edge has a weight that is a multiplication of road length, maximum speed, and the number of lanes.

In practical, due to the limitation of METIS, the original road network should be a strongly connected and contiguous graph. Therefore, we first had to extract the largest contiguous part of the raw road network. The result was a network with 770,192 intersections and 2,089,374 roads. This graph partitioning is conducted is a static manner before running simulation.

5.3 Distributing Agents Trip Data for Multi-Node Environment

As the number of agents increases, it takes much longer to read the data files about the agents including each agent's destination. If all of the nodes read all of the travel data file, the scalability will be limited by the data access. To obtain high scalability with larger numbers of nodes, we divided the file into smaller partial files for each node.

5.4 Performance Scalability on Supercomputers

We evaluated the scalability and acceleration with increasing numbers of supercomputer nodes for XAXIS simulations in the C++ backend compared to the Java backend. In this experiment, we simulated one million vehicles running for 10 minutes (600 simulation steps). As shown in Figure 4(a), the simula-

tion with Managed X10 and Native X10 achieves good scalability with increasing number of nodes. According to Figure 4(d), both of them achieves more than 10 times speedup with 256 nodes with comparison to 8 nodes. The ideal linear speedup is 32 times (256 divided by 8), but this is strong-scaling comparison so we need more computation loads when using large number of nodes such as 128 and 256 nodes. Figure 5 shows the comparison in memory usage between Managed X10 and Native X10, but obviously the memory footprint of Native X10 is much smaller than Managed X10. Figure 4(c) focuses more on the performance scalability of Native X10 and the simulation time is dramatically decreased from more than 800 seconds with 8 nodes to 78 seconds with 128 nodes to simulate 600 steps for the whole-country traffic simulation.

The main reason the simulation time with the C++ backend greatly differs from the Java backend is that the native C++ executable runs more efficiently than the Java virtual machine. Also the serialization and deserialization cost in Native X10 is much more lightweight than Managed X10. In addition, we can use MPI connections with the C++ backend, which takes greater advantage of the fast communication hardware of TSUBAME 2.0. Figure 4(e) shows the CPU usage but Native X10 obviously takes advantage of the CPU power.

5.5 Preliminary Performance Evaluation towards Billion-Scale Agent Simulation

After estimating the computation time and memory consumption, we initially simulated 100 million agents (1/10 of billion agents) with 128 and 256 nodes. The experimental condition is the same as in Section 5.4 with the C++ backend for the X10 runtime, 600 simulation steps, and the same computing environment. The elapsed times are shown in Figure 4(f).

According to Figure 4(a), the traffic simulations using one million trips show few differences between 128 and 256 nodes, since the numbers of agents are not sufficient to strain that many compute nodes. Figure 4(f) shows that the simulation time increases with the increasing number of trips on 128 and 256 nodes. With 128 nodes the elapsed time with 10 million agents simulation is 1.5 times larger than with 1 million agents and 8 times larger with 100 million trips. The same performance characteristics were observed for 256 nodes when comparing 1 million trips and 10 million trips. This means that communication overhead is significant when many agents are migrating among multiple nodes and the calculations of the running agents in each node could not mask this overhead. This suggests future work is needed to increase the number of agents towards a billion agents.

6 CONCLUSION AND FUTURE DIRECTION

In this paper, we described the design and implementation of Native X10-based high performance scalable simulations, and then showed how large-scale and high performance traffic simulation can be developed on top of such middleware. The experimental results demonstrated that only around 78 seconds were needed for 600 simulation steps for the national road network of Japan when using 1,536 CPU cores and 128 compute nodes, which ran faster than real time. As far as we know, our work as described in this paper is the first effort to design and implement multi-agent simulation middleware in the state-of-the-art PGAS language and the first large-scale experiments with such a system on a modern supercomputer.

Our future work will involve other applications of multi-agent simulations, such as viral marketing simulations, social network simulations, pandemic simulations, global economy simulations and so forth. To reach simulations with billions of agents, we will need to use more nodes. We believe that this work will lead to new scalability problems and new research fields.

ACKNOWLEDGMENTS

The authors would like to thank members of the transportation project including Takashi Imamichi, Hideyuki Mizuta, and Takayuki Osogami from IBM Research – Tokyo. The project is also partly supported by the JST CREST "Development of System Software Technologies for post-Peta Scale High Performance Computing".

REFERENCES

Karypis, G., and V. Kumar. 1998. "Multilevel k-way Partitioning Scheme for Irregular Graphs." *Journal of Parallel and Distributed Computing.* 48:96-129.

Nayar, S., R. Loffredo, C. Punyanitya, P. Weinstein, and D. Cashbaugh. 2005. "Large-Scale Multi-Agent-Based Simulation using Exemplars." *AAMAS*

Open Street Map. http://openstreetmap.jp/

Osogami, T., T. Imamichi, H. Mizuta, T. Morimura, R. Raymond, T. Suzumura, R. Takahashi, and T. Ide. 2013. IBM Mega Traffic Simulator. Technical Report, IBM Research - Tokyo, 2013

Paolucci, M., D. Kossman, R. Conte, P. Lukowicz, P. Argyrakis, A. Blandford, G. Bonelli, S. Anderson, S. de Freitas, B. Edmonds, N. Gilbert, M. Gross, J. Kohlhammer, P. Koumoutsakos, A. Krause, B. -O. Linnér, P. Slusallek, O. Sorkine, R. W. Sumner, and D. Helbing. 2012. "Towards a living earth simulator. No-vember 2012". *The European Physical Journal Special Topics*. Volume 214. Issue 1:77-108.

Parker, J. 2005. "A flexible, large-scale, distributed agent based epidemic model." *Winter Simulation Conference*

Perumalla, K. S., and S. K. Seal. 2012. "Discrete event modeling and massively parallel execution of epidemic outbreak phenomena". *SIMULATION: Transactions of the Society for Modeling and Simulation International*.

Saraswat, V. A., V. Sarkar, and C. von Praun. 2007. "X10: concurrent programming for modern architectures." *In Proceedings of the 12th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP '07)*. ACM. New York. NY. USA:271-271.

Šišlák, D., P. Volf, M. Jakob, and M. Pěchouček. 2009. "Distributed Platform for Large-Scale Agent-Based Simulations." *Springer*.

Suzumura, T., and H. Kanezashi. 2012. "Highly Scalable X10-based Agent Simulation Platform and its Application to Large-scale Traffic Simulation." *IEEE/ACM 16th International Symposium on Distributed Simulation and Real Time Applications (DS-RT 2012)*. Dublin, Ireland. October 25-27.

Suzumura, T., M. Takeuchi, S. Kato, T. Imamichi, H. Kanezashi, T. Ide, and T. Onodera, 2012. "X10-based Massive Parallel Large-Scale Traffic Flow Simulation." *PLDI 2012 X10 Workshop*

Takahashi, T. and H. Mizuta. 2006. "Efficient Agent-based Simulation Framework for Multi-Node Supercom-puters". *The Proceedings of the 2006 Winter Simulation Conference*.

Theodoropoulos, G., Y. Zhang, D. Chen, S. J. Turner, W. Cai, and Y. Xie. 2006. "Large Scale Distributed Simulation on the Grid." *CCGrid*

Yamamoto, G., H. Tai, and H. Mizuta. 2007. A Platform for Massive Agent-based Simulation and its Evaluation. *AAMAS*:900-902.

AUTHOR BIOGRAPHIES

TARO SUZUMURA is a visiting associate professor at the Graduate School of Information Science and Engineering of Tokyo Institute of Technology. He received his Ph.D. in Computer Science from Tokyo Institute of Technology in 2004. He joined IBM Research - Tokyo in 2004 and had involved with several projects such as high performance XML processing, the PHP scripting language, stream computing, the X10 programming language and so forth. Since 2010, he has started to develop an X10-based agent simulation platform and its application to large-scale traffic simulation. He is the a visiting associate professor since April 2009 and explores research on high performance computing and large-scale graph analytics. Since April, 2013, He also serves as a visiting associate professor at University College Dublin, Ireland. His e-mail address is toyo@jp.ibm.com.

HIROKI KANEZASHI is a student at the Graduate School of Information Science and Engineering of Tokyo Institute of Technology. He enrolled there in April 2013. His e-mail address is kanezashi.h.aa@m.titech.ac.jp.