IEICE TRANS. INF. & SYST., VOL.E94–D, NO.12 DECEMBER 2011

PAPER Special Section on Parallel and Distributed Computing and Networking

NSIM: An Interconnection Network Simulator for Extreme-Scale Parallel Computers

Hideki MIWA[†a)], **Ryutaro SUSUKITA**[††], ***Nonmembers***, **Hidetomo SHIBAMURA**[††], ***Member***, **Tomoya HIRAO**[††], **Jun MAKI**[††], **Makoto YOSHIDA**[††], **Takayuki KANDO**[†††], **Yuichiro AJIMA**[†], **Ikuo MIYOSHI**[†], ***Nonmembers***, **Toshiyuki SHIMIZU**[†], ***Member***, **Yuji OINAGA**[†], ***Nonmember***, **Hisashige ANDO**[††††], ***Member***, **Yuichi INADOMI**[†††††], ***Nonmember***, **Koji INOUE**[†††††], ***Member***, **Mutsumi AOYAGI**[†††††], ***Nonmember***, ***and*** **Kazuaki MURAKAMI**[†††††], ***Member***

SUMMARY In the near future, interconnection networks of massively parallel computer systems will connect more than a hundred thousands of computing nodes. The performance evaluation of the interconnection networks can provide real insights to help the development of efficient communication library. Hence, to evaluate the performance of such interconnection networks, simulation tools capable of modeling the networks with sufficient details, supporting a user-friendly interface to describe communication patterns, providing the users with enough performance information, completing simulations within a reasonable time, are a real necessity. This paper introduces a novel interconnection network simulator NSIM, for the evaluation of the performance of extreme-scale interconnection networks. The simulator implements a simplified simulation model so as to run faster without any loss of accuracy. Unlike the existing simulators, NSIM is built on the execution-driven simulation approach. The simulator also provides a MPI-compatible programming interface. Thus, the simulator can emulate parallel program execution and correctly simulate point-to-point and collective communications that are dynamically changed by network congestion. The experimental results in this paper showed sufficient accuracy of this simulator by comparing the simulator and the real machine. We also confirmed that the simulator is capable of evaluating ultra large-scale interconnection networks, consumes smaller memory area, and runs faster than the existing simulator. This paper also introduces a simulation service built on a cloud environment. Without installing NSIM, users can simulate interconnection networks with various configurations by using a web browser.

key words: *discrete event simulation, multiprocessor interconnection, parallel processing*

1. Introduction

The upcoming extreme-scale massively parallel computer systems should have about hundreds of thousands of nodes. Developing new communication libraries and porting existing applications to such large machines are challenging tasks. In order to build more sophisticated control schemes for future interconnection networks, and optimize communication algorithms and applications for such networks, it is necessary to investigate the performance degradation problems, such as the network congestion. The performance problems have to be solved before the development of the systems is completed. It is crucial to have a tool capable of simulating the transfer of packets or flits between nodes, the operation of switches, and the behavior of the communication functions. The abundance of the supported simulation patterns, the deriving information, the accuracy of the results and the time it takes for the complete simulation process are major issues to consider too.

In addition, it is difficult to predict the performance problems of such systems statically and qualitatively, because there are certain factors to take into account, such as the network contention and the load imbalance of the nodes. However, the performance prediction of applications via simulation can provide insights to help optimize applications so that they are ready to be run on the actual machines as soon as they become available. Even for existing large parallel machines, time for tuning applications on large machines as well as the queuing time before allocation of nodes can be very long. Hence, a tool that can evaluate the performance of interconnection networks, communication algorithms and applications is a real necessity. Therefore, the simulator presents a more available alternative so that minimal supercomputing time is consumed by debugging and performance optimization.

The existing simulators implement detailed models of target systems so that they can simulate accurately. However, it will be difficult for the simulators to complete a simulation of an extreme-scale alltoall communication within a reasonable time. The alltoall communication is indispensable for a fast Fourier transform required by a global atmospheric simulation and a molecular dynamics simulation. The total amount of data injected into the network by the alltoall communication is given by $O(n^2)$ or $O(n^2 \log(n))$ where n is the number of nodes. Thus, simulation time can be proportional to n^2 or $n^2 \log(n)$. It is not easy to keep the simulation time reasonable by increasing the number of nodes on the simulation environment, even if the existing simulators offer enough scalability. In order to complete simulations within a reasonable time, simulation should be accelerated.

In this paper, we propose a novel execution-driven parallel simulator for extreme-scale interconnection networks, NSIM. By simplifying a simulation model, this simulator

Manuscript received January 7, 2011.
Manuscript revised June 10, 2011.

[†]The authors are with Fujitsu Limited, Kawasaki-shi, 211–8588 Japan.
[††]The authors are with ISIT, Fukuoka-shi, 814–0001 Japan.
[†††]The author is with Qualiarc Technology Solutions Ltd., Tokyo, 166–0013 Japan.
[††††]The author was with Fujitsu Limited, Kawasaki-shi, 211–8588 Japan.
[†††††]The authors are with Kyushu University, Fukuoka-shi, 812–8581 Japan.
a) E-mail: miwa.hideki@jp.fujitsu.com
DOI: 10.1587/transinf.E94.D.2298

can run faster without sacrificing accuracy. NSIM offers a flexible and MPI-compatible interface called MGEN API. By using the MGEN API, users can describe communication patterns on a program (called a MGEN program) and correctly simulate it. Supported patterns are point-to-point communications, collective communications, and more complex patterns that are changed by the network congestion. This feature stems from the execution-driven simulator design. Because the message and packet is generated adaptively in accordance with the network simulation, this simulator can accurately simulates network-dependent communication patterns and successfully reproduce the congestion in the interconnection networks. The simulator also supports multiple NICs and a zero-copy communication by using the extended declarations of the MGEN API. NSIM can act as a performance measurement tool of interconnection networks for our previous proposal BSIM [1].

In addition, we explain a simulation service over the Internet. NSIM is modified to OpenNSIM so as to work on a cloud environment called a TaaS framework [2]. By accessing this service through a web browser, users can simulate interconnection networks without installing the simulator.

The rest of this paper is organized as follows. In Sect. 2, related work in existing interconnection simulators is presented. In Sect. 3, we go into the details of NSIM by revealing the inputs, outputs, functions, and implementation. In Sect. 4, we introduce a simulation service via a cloud environment. In Sect. 5, the simulation accuracy and the capability of simulating extreme-scale interconnection networks are confirmed by the experimental evaluations. In addition, an example of usage by the simulation service is described. In Sect. 6, we conclude our paper.

2. Related Work

There are many good simulators of large-scale interconnection networks in the literature. BlueGene/L interconnection network simulator, proposed by N.R. Adiga et al., is a trace-driven simulator [3]. This simulator specializes in BlueGene/L and is used for the performance prediction and the analysis of the interconnection network. Traces are generated by the IBM's trace capture utility based on a pseudo code. This simulator employs the shared-memory parallel simulation approach. It runs on a relatively small machine such as a 16-way IBM POWER3+ SMP node with 64 GB memory. Several threads are concurrently executed at runtime. In order to synchronize target simulation time among threads, YAWNS (yet another windowing network simulator) protocol is used [4]. This is a conservative parallel simulation protocol.

BigNetSim, introduced by L.V. Kale et al., is a parallel simulator of interconnection networks [5]. This simulator supports detailed network models of various topologies. Many configuration parameters of networks, such as topology, network sizes and latency, are available. This simulator is built on the POSE, a general-purpose optimistically-synchronized PDES (parallel discrete event simulation) environment [6]. This programming system supports the process virtualization. Helped by this, BigNetSim can exploit the large parallelism of the simulation execution environment around one hundred of processes. This simulator has two running modes. The first mode is driven by the artificial traffic patterns generated by internal traffic generators. The second mode is a trace-driven simulation mode. In this mode, the simulator requires trace files generated by BigSim [7]. BigSim is a parallel simulator for extremely large parallel machines. It has a simple interconnection network model, which does not consider network contention. A target program should be linked to the Charm++ library [8]. BigSim is implemented inside the Charm++ library. On the execution of the program, BigSim generates the trace files. BigNetSim uses them as inputs, and executes the interconnection network simulation based on the detailed network model. After simulation, many kinds of statistics can be displayed graphically. For example, the CPU usage statistics, the breakdown of the process time, the transferred message size, and so on. These are very useful for the application tuning.

FSIN is a functional interconnection network simulator, which is included in a simulation framework INSEE [9]. This simulator does not support the detailed network simulation, and cannot be executed in parallel. It assumes that the event processing time takes one FSIN cycle for every network events. SICOSYS [10] is also an interconnection network simulator from the INSEE framework. This simulator implements the detailed network model. It can only simulate up to one thousand nodes within a practical time. The INSEE framework has a distinguishing characteristic. It includes the feedback mechanism from interconnection network simulator to the trace generator, called TrGen [11].

SMART is a general-purpose sequential simulator for parallel architectures [12]. This simulator adopts a unique simulation methodology, process-driven simulation. In this method, the target system is implemented as a set of communicating processes. Each process is corresponding to a hardware device. Processes communicate by exchanging information through message passing and accessing shared variables. The advantage of this method is flexibility and modularity. The disadvantage is that the simulation speed is slower than execution-driven simulation.

3. NSIM

3.1 Simulation Model

We have modeled one, two, three and higher dimensional torus/mesh networks and a full bisectional bandwidth (FBB) fat-tree network with two or three layers. One example of the higher dimensional torus/mesh network is Tofu, which has a six-dimensional topology [13]. The node and the router are connected by bi-directional links.

The node has one CPU and main memory that contains two buffers, the USER buffer and the MPI buffer. We assume that the only one thread is executed on the CPU.

The USER buffer represents a set of all the memory regions allocated by users in the MGEN program. It contains the sending and the received data. In the two-copy communication, the data is transferred by way of the MPI buffer. In the zero-copy communication, the MPI buffer is not used.

In the router model, we assume the static dimension-ordered routing, the virtual cut-through and the pipelined router. The router has input queues, a crossbar switch and output buffers. The data transfer in the network is modeled in a packet level. A flit level can achieve better accuracy than the packet level. However, simulating in the flit level is so slow that the simulator cannot complete the extreme-scale simulation within a reasonable time. Without employing the flit level, simulating virtual cut-through networks in the packet level can achieve the same accuracy as in the flit level. This simplification dramatically reduces the simulation events, and helps the simulator to run fast without any loss of accuracy.

In order to show that simulating virtual cut-through routing in the both levels has comparable accuracy, we describe the transfer of the packet in the flit-level as follows. We assume that a current switch forwards a packet from a previous switch to a next switch.

1. When a head flit of the packet arrives at the current switch, the switch checks if there is enough space left in the receive buffer on the next switch by using a credit-based flow control mechanism.

2. The current switch starts to forward the first flit of the packet after the next switch accepts a packet. Then, a successor flit is sent to the next switch after receiving the flit from the previous switch and sending a precedent flit to the next switch.

3. The packet transfer is finished if the last flit is arrived at the next switch.

We define that the current switch w receives a flit k ($0 \leq k < N_p$) from the previous switch at time $T_r(w,k)$ and sends it to the next switch at time $T_s(w,k)$ where N_p is the number of flits included in a packet p. By using these notations, $T_s(w,k)$ can be expressed as follows.

$$\begin{cases} T_s(w,k) \geq T_r(w,k) & (k=0), \\ T_s(w,k) = max(T_s(w,k-1) + F/b, T_r(w,k)) & (k>0). \end{cases}$$

In the equation, F and b represents the size of a flit and the bandwidth of a switch. We assume that the network consists of the same switches and the same network interfaces, while the throughput between a switch and a network interface is not necessarily identical. In addition, we define that $max(X,Y)$ returns the largest value from the numbers X, Y. Then, there can be four cases for the second equation as described below.

case 1) $T_s(w,k) = T_s(w,k-1) + F/b$ for $N_p > k > 0$

case 2) $T_s(w,k) = T_r(w,k)$ for $N_p > k > 0$

case 3) $T_s(w,k) = T_s(w,k-1) + F/b$ for $j \geq k > 0$, and $T_s(w,k) = T_r(w,k)$ for $N_p > k > j$

case 4) $T_s(w,k) = T_r(w,k)$ for $j \geq k > 0$, and $T_s(w,k) = T_s(w,k-1) + F/b$ for $N_p > k > j$

In the case 1, each flit arrives at the same speed as the switch sends a flit. $T_s(w, N_p - 1)$ can be inductively calculated and is equal to $T_s(w,0) + (N_p - 1) \cdot F/b$. In the case 2 and 3, each flit arrives at the slower speed, because the bandwidth of the network interface is lower than the switch. In the case 2, a switch forwards a packet after the head flit of the packet arrives. On the other hand, in the case 3, the current switch cannot start to forward some flits until the next switch accepts a packet. In these two cases, $T_s(w, N_p - 1)$ is equal to $T_r(w, N_p - 1)$. The case 4 cannot be occurred except for technical trouble of switches. Thus, we ignore this case. Since $T_s(w,0)$ depends on $T_r(w,0)$ and the status of the next switch, $T_s(w, N_p - 1)$ can be obtained if $T_r(w,0)$ and $T_r(w, N_p - 1)$ are known. Therefore, we need to simulate the first and the last flits of each packet except for intermediate flits.

In order to build the packet-level simulation model, we define a packet-level event so that the event has a meaning of the arrival of a head flit of a packet in a switch and additional information of the arrival time of a tail flit. By using this model, the packet-level simulation of the virtual cut-through routing can have similar accuracy as the flit-level simulation.

3.2 Simulator Overview

Figure 1 depicts the simulator overview. This figure outlined three major parts, the inputs on the left, the body of the simulator in the middle, and the outputs on the right side. The inputs are the parallel program (MGEN program) that describes target traffic patterns to evaluate and the simulator configuration including the mapping between target processes and nodes. The simulator is initialized based on the configuration. In order to simulate the different types of machine, users have to change the configuration file. The rank map helps the software developers to evaluate topology-aware applications and communication algorithms.

The simulator provides a MPI-compatible programming interface called MGEN API. By using the API, users can describe communication patterns for the simulator in the same way that they write MPI programs. We named such program a MGEN program. The MGEN program is intended to describe the communications in the target system. MGEN programs can be generated from the original MPI programs by the following instructions.

1. Change the prefixes of every declaration from MPI_ to MGEN_.

2. Replace the name of the main function with MGEN_Main.

3. Delete the undefined function calls, such as MGEN_Init,

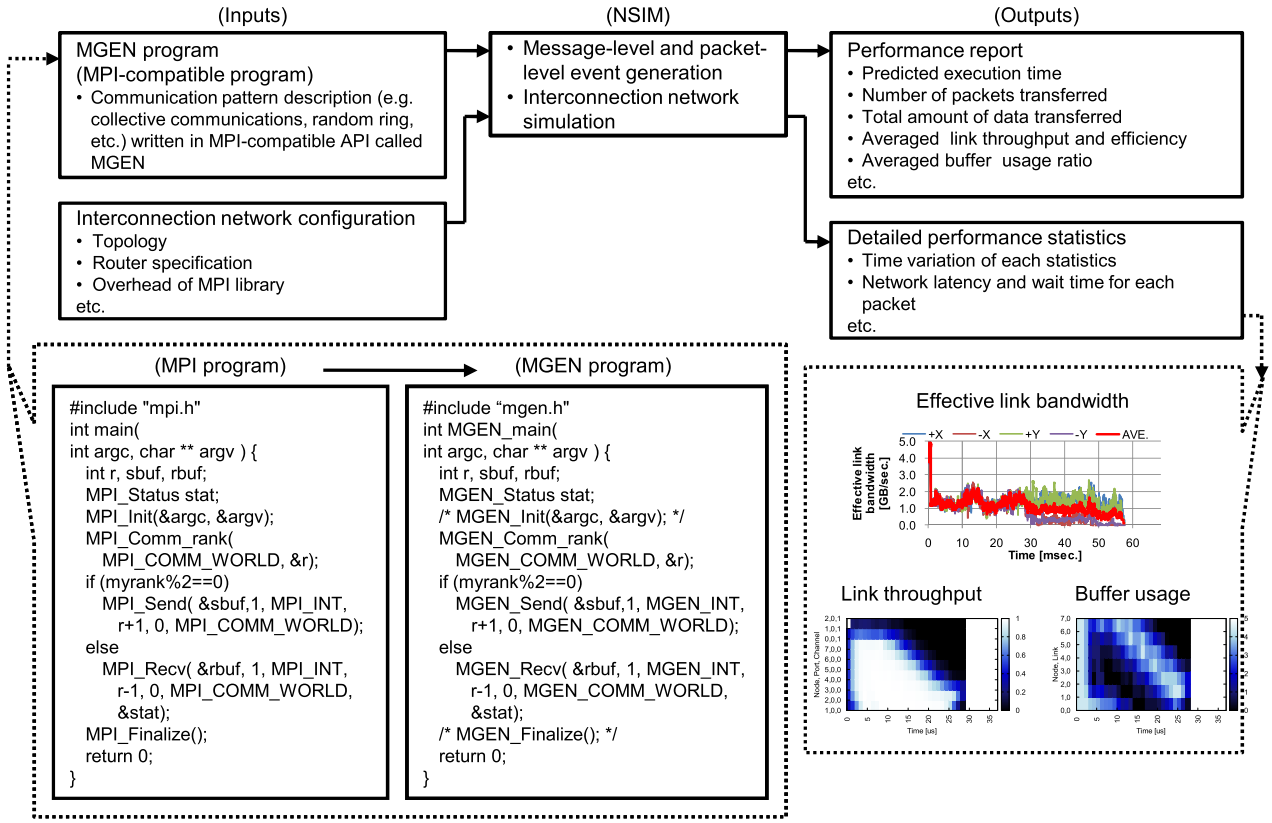

Fig. 1 NSIM inputs and outputs.

MGEN_Finalize, because the simulator does not provide them.

4. Replace the computation kernels with MGEN_Comp(t) function calls, if the users want to include the computation time to the simulation result.

In order to manipulate the multiple NICs and support the zero-copy communication, NSIM offers the LMGEN_* functions. Figure 2 shows the sample code. In this example, each process sends messages to two neighbor processes and receives messages from them. Since there is no argument to specify the NIC id or the message-copy mode in the MGEN_Isend and MGEN_Irecv functions, we add appropriate arguments to new functions LMGEN_Irecv and LMGEN_Isend. The first added argument represents the id of the NIC, while the second added argument is the switch of the zero-copy communication.

There are tree constraints of the MGEN program. The first constraint is that the MGEN API does not provide all the MPI-compatible declarations. For example, MGEN_Comm_split is not defined. The second constraint is that the functions in the MGEN API do not perform real communications. Therefore, it is not possible to write the MGEN programs that depend on the values in the received messages. This limitation largely contributes to the reduction of memory consumption by the simulator. The third constraint is that MGEN functions can only affect the processing time of each target process. By using a special function of the MGEN API, the processing time of computation kernels can be included in the time of each target process. Computation time should be specified as the argument in the function. This function advances time in the target system. We recognize that these constraints are not acceptable to some interconnection network researchers. Currently, users

```
#include "mgen.h"
#include <string.h>
int MGEN_Main(int argc, char ** argv) {
  int rank, size, ms = 4, r, l;
  MGEN_Request rq[4];
  MGEN_Status st[4];
  MGEN_Datatype dt = MGEN_BYTE;
  MGEN_Comm com = MGEN_COMM_WORLD;
  t_TsimRoutingInfo i0,i1;
  bzero(&i0,sizeof(t_TsimRoutingInfo));
  bzero(&i1,sizeof(t_TsimRoutingInfo));
  MGEN_Comm_rank(com, &rank);
  MGEN_Comm_size(com, &size);

  r = (rank + 1) % size;
  l = (rank - 1 + size) % size;
  i0.SrcNIC = i0.DstNIC = 0;
  i1.SrcNIC = i1.DstNIC = 1;

  LMGEN_Irecv(NULL, ms, dt, r, 0, com, &rq[0], &i0, 1);
  LMGEN_Irecv(NULL, ms, dt, l, 1, com, &rq[1], &i1, 1);
  LMGEN_Isend(NULL, ms, dt, r, 1, com, &rq[2], &i1, 1);
  LMGEN_Isend(NULL, ms, dt, l, 0, com, &rq[3], &i0, 1);

  MGEN_Wait(&rq[0], &st[0]); MGEN_Wait(&rq[1], &st[1]);
  MGEN_Wait(&rq[2], &st[2]); MGEN_Wait(&rq[3], &st[3]);
  return(0);
}
```

Fig. 2 Sample MGEN program using LMGEN functions.

are encouraged to describe communication patterns so as not to be affected by the constraints if possible.

In the simulator configuration, users can specify the parameters, e.g. the MPI latencies, the specification of the routers, the specification of the network. The information contained in the configuration includes the statistics file name, the log file name, the router specification, the mapping file name, the network topology, the overheads of processing MPI functions, etc. Users can define arbitrary mappings between the target processes and the nodes by describing the mapping between the rank of a target process and the address of a node.

This simulator can generate two kinds of information, the performance report and the detailed performance statistics. The performance statistics include the execution time predicted for the MGEN program, the effective bandwidth of the link, the efficiency of the link, the cumulative size of the data transferred, etc.

The detailed performance statistics includes the throughput of each link, the waiting time of the packets in each node, the occupancy rate of each buffer, the communication latency of each message, the number of the packets that has the longer residence time in buffers than the specified period, etc. The statistics is collected every period for each node. By using tools supplied with NSIM, the statistics can be displayed graphically, as shown in Fig. 1 (at the bottom right). Users can confirm or analyze state of communications between routers or processes for each period of time.

3.3 Simulation Flow

In this subsection, we give a full detail of each function module. The simulator is built on the execution-driven simulation approach, because this method can simulate correctly the dynamic behavior in the target system, e.g. the dynamic load balance, the network congestion. For example, the dynamic load balancing application performs this kind of communications. In this application, a master process receives a message from one of worker processes, and assigns certain jobs to it. These assignments depend on the network congestion. It is difficult for trace-driven simulation to support the communication patterns that are dynamically changed by the network congestion.

Figure 3 illustrates simulator organization. The simulator has five function modules, e.g. the MGEN, the PGEN, the SIM, the DES, the EP modules, and three event queues. The simulator handles two types of the events, the MLEs (message level events) and the PLEs (packet level events). The MLE is corresponding to a message in the message passing programming model. This event is composed of the source rank, the destination rank, the size of the message, etc. The PLE represents a packet in the interconnection networks of the target system. This event is made up of the source node address, the destination node address, the size of the packet, the number of flits in the packet, etc. The modules work in a coordinated manner as follows.

1. MLE generation: The SIM module checks if PLEs are in the PLE queue. If not found, the module requests the PGEN to generate PLEs. The PGEN checks if MLEs are left in the MLE queue. If found, the module generates PLEs from the MLE(s). If no events in the queue, the module requests the MGEN module to generate MLEs. The MGEN module generates the MLEs by executing the MGEN program, and put them in the MLE queue until the queue is full. If the module reaches a receive function or a wait function corresponding to a non-blocking receive function in the MGEN_Main, it also stops generating events.

2. PLE generation: The PGEN module pops an event from the MLE queue, and extracts PLEs from it. The module also determines the time to process the PLEs based on the current time of the target node and the latency in the node.

3. Interconnection network simulation: The SIM module obtains PLEs from the PLE queue and passes them to the DES queue module if the event indicates packet transfer. The DES module puts the events to the DES queue. Then, the module requests the EP module to process the event. The EP module treats the PLE as a packet in the target system and carries out interconnection network simulation. The EP module checks the availability of network resources that are needed for transferring a packet in the target system. If available, the module calculates the latency, reserves the resources, and generates a new event that releases them. If the event does not indicate a packet arrival, the EP module generates a new event that represents the arrival of the packet in the next node and sends it to the DES queue. Otherwise, the EP module passes the packet to the SIM module. The SIM module checks if tail packets of messages are arrived. If the module detects the tail packet arrival, it notifies the MGEN module via the PGEN module that the message is arrived. The MGEN module restarts event generation.

3.4 Implementation

We implement the interconnection network simulation part based on the parallel discrete event simulation (PDES). The PLEs generated by the PGEN module are treated as the discrete simulation events in the interconnection network simulation part. We employed a conservative algorithm of the PDES instead of the optimistic one. In terms of the simulation speed, we cannot tell which algorithm is faster. From the design perspective, the conservative algorithm is easier to implement. Since this algorithm does not permit the speculative processing of the events in contrast to the optimistic one, no rollback mechanism is required.

In order to implement NSIM as the parallel program, we used the MPI, because a MPI program can run on a variety of parallel machines, e.g. multi-core workstations, PC

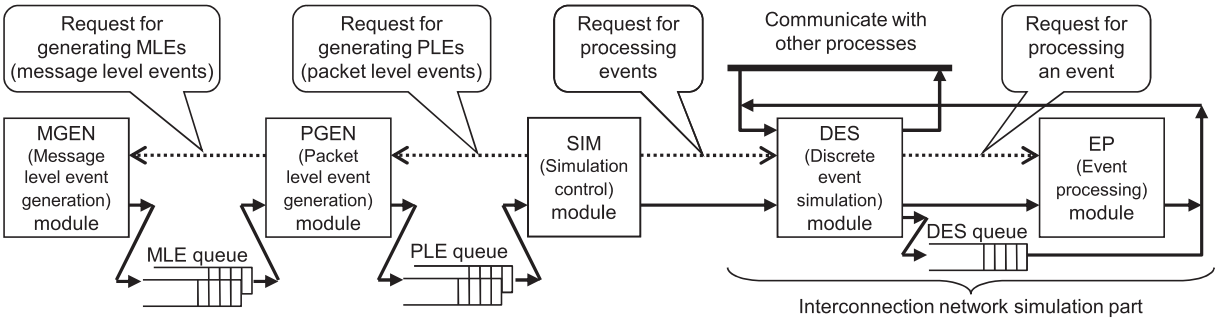

Fig. 3 NSIM simulation flow.

clusters, massively parallel computer systems.

4. A Simulation Service: TaaS+OpenNSIM

In order to make NSIM public, we decided to provide a simulation service over the Internet instead of distributing source files. By deploying the simulation service, people who have an internet connection can use NSIM at any time without installing the simulator on their computers. We have made changes to NSIM so that the simulator works on a cloud environment called TaaS (Tools as a Service) framework. We named the modified version of the simulator OpenNSIM.

The system consists of two parts, the front-end gui, and the TaaS framework. We describe each step blow.

1. By using a web browser, a user connects to the web site managed by the TaaS framework and submits an e-mail address and a one-time password.
2. The TaaS framework sends an e-mail including an URL which is unique to each run. When the user accesses the URL, the TaaS framework authenticate the user based on the one-time password and transfer the Java applet to the browser. The applet generates the front-end gui. The user can specify appropriate parameters and files through this interface. After all the parameters and files are set, the applet sends them back to the TaaS framework.
3. The TaaS framework receives them, confirms that the parameters are valid, and initiates OpenNSIM. When the simulation has successfully completed, the TaaS framework visualizes the results, compresses the files to a ZIP file, and notify the user of completion via an e-mail.
4. The user can get results from the URL supplied in the e-mail.

5. Experimental Evaluation

In this section, the two kinds of fundamental characteristics of this simulator are presented. First, the prediction accuracy is introduced. Then, the simulation performance is measured. Then, we explain the usage of the simulation service by an example.

5.1 Simulation Accuracy of NSIM

In order to examine the accuracy of the simulation, we compare the bandwidth of a random ring traffic between the simulation and the measurement on the real machine. The benchmark program is included in the HPC Challenge Benchmark suite [14]. For measuring bandwidth of parallel communication, all processes are arranged in a ring topology and each process sends and receives a 2 MB message from its left and its right neighbor in parallel.

We selected the Intel Endeavor cluster for comparison. The machine has two quad-core Intel Xeon X5560 CPUs running at 2.8 GHz in each node and connects up to 256 nodes by InfiniBand QDR switches. The switch provides unidirectional throughput of 4.0 GB/s and port-to-port latency of less than 100 nanoseconds, and supports adaptive routing. The parameters for the simulation are listed in Table 1. The values except for the MPI latency are determined in reference to the specification of the target system. The MPI latency is derived from the pingpong latency of the HPCC benchmark, since the value cannot be estimated exactly. However, the MPI latency is much shorter than the 1-hop latency of a 2 MB message and must have a small impact on the result in this experiment.

There are some differences between the simulation and the measurement on the real machine. The first difference is the benchmark program. The MGEN program is generated by extracting the program codes relevant to the random ring traffic. We modified the MGEN program so as to emulate multiple processes in each node, because the simulator supports one process per node. In addition, we also omitted the communications within a node, since the simulator does not support shared memory communication. The other difference can be the routing algorithm, the arbitration algorithm, and so on. These differences lead to errors in the simulation result.

We executed the MGEN program and calculated the bandwidth in the same way as the original benchmark (i.e. the geometric mean of ten different randomly chosen process orderings in the ring).

Figure 4 shows the results of the two cases, the predic-

Table 1 Configuration parameters for NSIM.

Type	Parameter	Value
Router	Unidirectional link throughput	4.0 GB/s
	Switch throughput	4.0 GB/s
	Routing computation time	4.0 ns
	Virtual-channel allocation time	4.0 ns
	Switch allocation time	4.0 ns
	Switch latency	90 ns
	Cable latency	10 ns
Node	DMA transfer rate	6 GB/s
	Memory bandwidth	6 GB/s
	MPI overhead	1.6 us
	Number of process	1 process/node

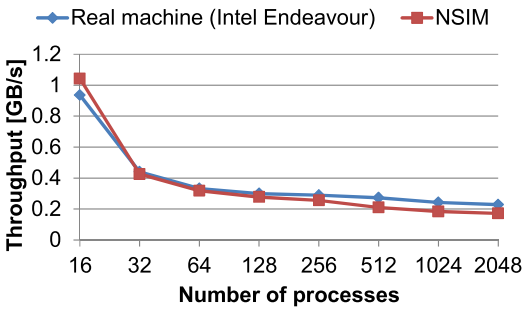

Fig. 4 Random ring bandwidth.

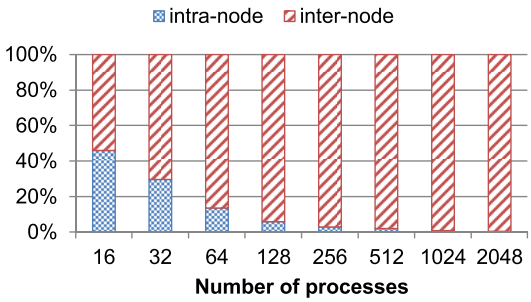

Fig. 5 Ratio of intra-node to inter-node communications.

tion by NSIM and the observed result on the cluster. The horizontal axis represents the number of nodes. The vertical axis stands for the bandwidth of the random ring traffic. They have the similar trends along the number of nodes. However, the gap exists between two results. There are two reasons for this gap. The first reason is that we removed the communications within a node by modifying the benchmark program. Figure 5 indicates the ratio of intra-node communications to inter-node communications. For a small number of processes, the intra-node communications account for a half of total communications. Although communicating with other processes on the same node via the shared memory can be much faster than on different nodes, the elimination of communication makes the predicted bandwidth higher. The second reason is that the routing algorithm and the arbitration algorithm can differ. For a large number of processes, the network congestion is likely to occur. Due to the differences, more serious congestions can be produced on the simulation compared to the real machine. Hence, the predicted bandwidth becomes lower than the observation on the real machine.

5.2 Simulation Performance Evaluation of NSIM

In order to explore the capability of evaluating the extreme-scale interconnection networks on a small machine such as workstations, we have evaluated the simulation time and the maximum consumed memory size of NSIM . In addition, in order to show the superiority in a small machine, we also compare to the existing simulator, BigNetSim, in the same environment. We used BigNetSim Rev. 11877 and the corresponding version of Charm++ library (i.e. BigSim) from the cvs repository.

The Dell Precision T7400 workstation is used for a simulation environment. The specification of the machine is listed in Table 2. Each simulator runs on eight processor cores. The simulation time is taken from the simulator reports that are derived at the end of the simulations. The execution time of BigSim is not included into the run time of BigNetSim, because it is relatively short and can be ignored. The maximum memory usage is observed by using ps command and summarizing the virtual memory size consumed by the eight processes. We execute ps command with '-o vsz' argument at every 0.01 seconds in the first one second, then every 1 second after that.

We use the Bruck's alltoall algorithm [15] as the benchmark program. We utilize the MPI program for BigSim that is the trace generation program for BigNetSim. We have evaluated the communication with the different message sizes, 4 Btyte and 1024 Byte.

Figure 6 shows the execution time of the simulators. We could obtain the results up to 8x8x4 nodes for BigNetSim, and up to 16x16x8 (for the 1024 Byte message) and 64x32x32 (for the 4 Byte message) nodes for NSIM. We confirm that NSIM is at least three hundred times as fast as BigNetSim in this experimental environment. The results also show that NSIM has the capability to simulate extreme-scale systems that have nearly one hundred thousands of nodes. This simulator can work faster than BigNetSim, because the simulation methodology differs. While NSIM is an execution-driven simulator, BigNetSim is a trace-driven simulator. BigNetSim analyzes dependency between simulation events and reorders them so as not to produce incorrect results. Meanwhile, NSIM does not need to track the dependency of simulation events.

Figure 7 shows the maximum consumed memory size for both simulators. NSIM could run with the smaller memory area than BigNetSim. The consumed memory size of BigNetSim is about eight times as large as that of NSIM.

The above results indicate that NSIM can evaluate future interconnection networks on a desk-side workstation. There will be new merits for users. For example, by processing the logs along with the simulation, the simulator can help users observe the network congestion in the target system before the completion of the simulation. Since

Table 2 Specification of the simulation environment.

Parameter	Setting
Machine	Dell Precision T7400 workstation
CPU	two Quad-Core Intel Xeon (R) processors E5440 2.8 GHz with 2x6 MB L2 cache
Memory	32 GB quad-channel DDR2-SDRAM memory 667 MHz with ECC
OS	Red Hat Enterprise Linux 5
Compiler	gcc version 4.1.2
MPI library	MPICH2 1.0.8 [16]

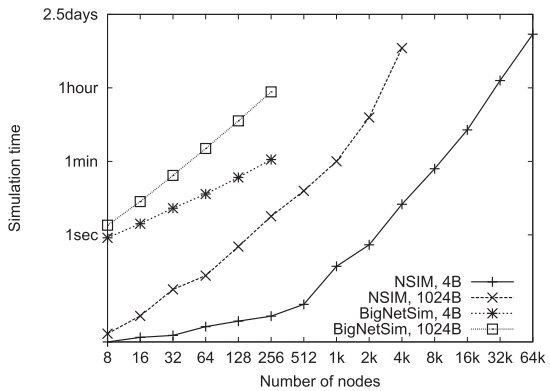

Fig. 6 Simulator execution time of NSIM and BigNetSim.

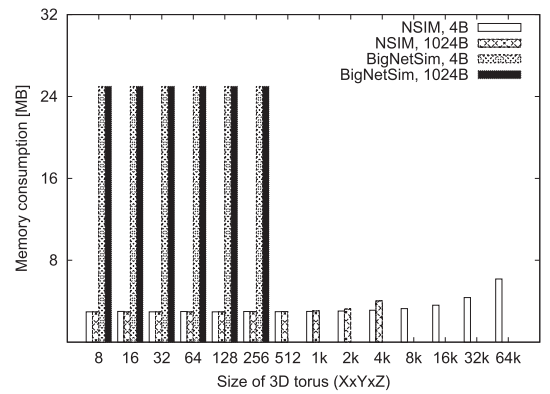

Fig. 7 Maximum consumed memory size.

the simulation time for the large-scale networks tends to be longer, providing such information during the simulation is effective in reducing the performance evaluation period.

5.3 Usage Example of the Simulation Service

In this section, we describe the usage of the simulation service by an example. We evaluate alltoall communication implemented by the pairwise exchange algorithm on a 3D torus network which contain 2048 nodes in an 16x16x8 structure. The other parameters are listed in Table 3. The specification of the target system is determined based on the state-of-the-art supercomputers. We used the Dell Precision T7400 workstation as a back-end server. It takes twenty four minutes to complete the simulation. The predicted communication time is 166.7 ms. Results are visualized.

Table 3 Specification of the simulation environment.

Parameter	Value
Routing	dimension ordered with dateline
Packet transfer	Virtual cut through
Unidirectional link bandwidth	4 GB/s
Routing calculation	4 ns
Virtual channel allocation	4 ns
Switch allocation	4 ns
Switch transfer	4 ns
Switch latency	78 ns
Cable latency	10 ns
MTU	2 KiB
Packet length	32 B - 2 KiB (MTU)
Packet header	32 B
Number of virtual channels	2
Virtual channel buffer	8 KiB (MTU×4)
Flit length	16 B
DMA bandwidth	16 GB/s
Memory bandwidth	16 GB/s
MPI overhead	200 ns

Figure 8 and 9 shows time variation of a direction-averaged link throughput for six directions (plus or minus in X, Y, or Z axis), and network latency. The horizontal axis represents elapsed time from the beginning of the simulation. The vertical axis stands for a link throughput in GB/s (Fig. 8), and the latency in ps (Fig. 9).

According to Fig. 8 and 9, there are peak or bottom points for each direction. This phenomena can be explained by a communication pattern of the pairwise exchange algorithm on the three-dimensional torus network. This algorithm has p communication phases where p is the number of nodes. In each phase, the algorithm determines new pairs of nodes that exchange messages. When one of the coordinate values of the Hamming distance between nodes belonging to the same pair is equal to half of the axis size, every message goes to the same direction in the axis and shares links. Thus, link throughput and network latency have increased at the same. The above situation occurs $y \cdot z$, $x \cdot z$, and $x \cdot y$ times in the x, y, z-axis where x, y and z are the size of each axis. After excluding continuous occurrences, there are $y \cdot z$, z, and 1 times for each axis. This pattern matches the result in Fig. 8.

By using OpenNSIM, we can observe an internal behavior of interconnection network of the target system. Therefore, this simulator can be used for debugging and tuning programs in addition to performance evaluation.

6. Conclusions and Future Work

In this paper, we have introduced an execution-driven simulator, called NSIM, in order to predict the performance of extreme-scale interconnection networks. Unlike conventional trace-driven approaches, NSIM makes it possible to handle communication patterns that are changed dynamically by the network congestion. This simulator provides the flexible and MPI-compatible interface to describe a variety of communication patterns, including the network-

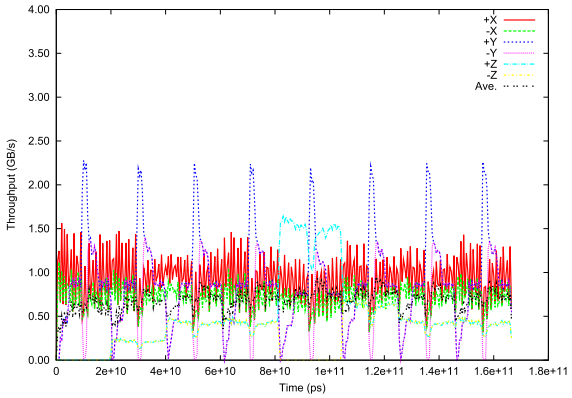

Fig. 8 Time variation of link throughput.

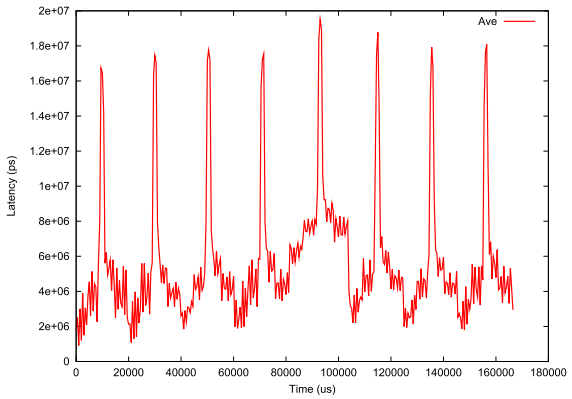

Fig. 9 Network latency.

dependent communication patterns. Experimental result shows that the simulator can accurately predict the performance trends of the real machine. We also verified that this simulator is capable of simulating future extreme-scale interconnection networks that have nearly one hundred thousands of nodes. We also introduced OpenNSIM, an interconnect simulation service via a cloud environment. By providing NSIM as a service, users can simulate many kinds of interconnection networks without downloading, building, installing and upgrading the simulator.

Although we do not present in this paper, this simulator can work as a performance measurement tool of interconnection networks for our previous proposal BSIM [1]. We are performing the cooperative simulation by using BSIM and NSIM to show the capability of simulating future systems. We are also working on removing the limitations of the MGEN program, implementing other topologies and routing algorithms, and preparing debugging tools for MGEN programs.

References

[1] R. Susukita, H. Ando, M. Aoyagi, H. Honda, Y. Inadomi, K. Inoue, S. Ishizuki, Y. Kimura, H. Komatsu, M. Kurokawa, K.J. Murakami, H. Shibamura, S. Yamamura, and Y. Yu, "Performance prediction of large-scale parallell system and application using macro-level simulation," Proc. 2008 ACM/IEEE Conference on Supercomputing, SC '08, Piscataway, NJ, USA, pp.20:1–20:9, 2008.

[2] "OpenNSIM" https://ngarch.isit.or.jp/taas/opennsim/

[3] N.R. Adiga, M.A. Blumrich, D. Chen, P. Coteus, A. Gara, M.E. Giampapa, P. Heidelberger, S. Singh, B.D. Steinmacher-Burow, T. Takken, M. Tsao, and P. Vranas, "Blue gene/l torus interconnection network," IBM J. Res. Dev., vol.49, pp.265–276, March 2005.

[4] D.M. Nicol, C.C. Michael, and P. Inouye, "Efficient aggregation of multiple pls in distributed memory parallel simulations," Proc. 21st Conference on Winter Simulation, WSC '89, pp.680–685, New York, NY, USA, 1989.

[5] N. Choudhury, Y. Mehta, T.L. Wilmarth, E.J. Bohm, and L.V. Kale, "Scaling an optimistic parallel simulation of large-scale interconnection networks," Proc. 37th Conference on Winter simulation, Conference, WSC '05, pp.591–600, 2005.

[6] T.L. Wilmarth and L.V. Kale, "Pose: Getting over grainsize in parallel discrete event simulation," Proc. 2004 International Conference on Parallel Processing, ICPP '04, pp.12–19, Washington, DC, USA, 2004.

[7] G. Zheng, G. Kakulapati, and L.V. Kalé, "Bigsim: A parallel simulator for performance prediction of extremely large parallel machines," Parallel and Distributed Processing Symposium, International, vol.1, p.78b, 2004.

[8] L.V. Kale and S. Krishnan, "Charm++: A portable concurrent object oriented system based on C++," Proc. Eighth Annual Conference on Object-Oriented Programming Systems, Languages, and Applications, OOPSLA '93, pp.91–108, New York, NY, USA, 1993.

[9] F.J. Ridruejo and J.M. Alonso, "Insee: An interconnection network simulation and evaluation environment," Proc. 11th Euro-Par Parallel Processing Conference 2005, Euro-Par'05, pp.1014–1023, SpringerLink, 2005.

[10] V. Puente, J.A. Gregorio, and R. Beivide, "Sicosys: An integrated framework for studying interconnection network performance in multiprocessor systems," Proc. 10th Euromicro Conference on Parallel, Distributed and Network-Based Processing, EUROMICRO-PDP'02, pp.15–22, Washington, DC, USA, 2002.

[11] F.J. Ridruejo, A. Gonzalez, and J.M. Alonso, "Trgen: A traffic generation system for interconnection network simulators," International Conference on, Parallel Processing Workshops, pp.547–553, 2005.

[12] F. Petrini and M. Vanneschi, "Smart: A simulator of massive architectures and topologies," International Conference on Parallel and Distributed Systems Euro-PDS'97, 1997.

[13] Y. Ajima, S. Sumimoto, and T. Shimizu, "Tofu: A 6D mesh/torus interconnect for exascale computers," Computer, vol.42, pp.36–40, Nov. 2009.

[14] P.R. Luszczek, D.H. Bailey, J.J. Dongarra, J. Kepner, R.F. Lucas, R. Rabenseifner, and D. Takahashi, "The HPC challenge (HPCC) benchmark suite," Proc. 2006 ACM/IEEE Conference on Supercomputing, SC '06, New York, NY, USA, 2006.

[15] J. Bruck, C.T. Ho, S. Kipnis, and D. Weathersby, "Efficient algorithms for all-to-all communications in multi-port message-passing systems," Proc. Sixth Annual ACM Symposium on Parallel Algorithms and Architectures, SPAA '94, pp.298–309, New York, NY, USA, 1994.

[16] "MPICH2." http://www-unix.mcs.anl.gov/mpi/mpich2/

Hideki Miwa received the B.E. and M.E. degrees in engineering from Kyushu University, Japan in 2002 and 2004, respectively. In 2007, he joined the Institute of Systems, Information Technologies and Nanotechnologies and worked on large-scale interconnection network simulations. In 2010, he moved to Fujitsu Limited. He has been engaged in the performance analysis of high performance computer systems. He is a member of the IPSJ.

Ryutaro Susukita received a Ph.D. degree in science from Kyoto University in 1997 and then joined RIKEN to work on special purpose computers for molecular dynamics. He also studied computational techniques for large-scale simulations in physics and astronomy using the special purpose computers. At present, he is working on large-scale simulations including NSIM and parallel discrete event simulations at the Institute of Systems, Information Technologies and Nanotechnologies. He is a member of the Physical Society of Japan and the Astronomical Society of Japan.

Hidetomo Shibamura received his B.E. and M.E. degrees in computer science and systems engineering from Kyushu Institute of Technology in 1992 and 1994. He was a research associate at the Center for Microelectronic Systems, Kyushu Institute of Technology from 1994 to 1998, and was a research associate of computer science at Kumamoto University from 1998 to 2006, where he received D. Eng. degree in computer science in 2006. He is a researcher at Institute of Systems, Information Technologies and Nanotechnologies since 2006. His research interests include high-performance computing, computer architecture, and VLSI system design. He is a member of IPSJ, IEEE and the IEEE Computer Society.

Tomoya Hirao was born in Hyogo prefecture, Japan in 1977. He received the B.E. from Wakayama University, Japan in 2000 and M.E. from Nara Institute of Science and Technology, Japan in 2002. In 2009, he joined the Institute of Systems, Information Technologies and Nanotechnologies. Currently, He is a research fellow of the Department of Advanced Information Technology, Kyushu University, Japan.

Jun Maki was born in Hokkaido, Japan in 1967. He received the Ph.D. degree in chemistry from Hokkaido University, Japan in 1999. Currently, he is a research fellow of the Institute of Systems, Information Technologies and Nanotechnologies (ISIT). His research interests are quantum chemistry, computational chemistry, and high-performance computing. He is a member of the Chemical Society of Japan.

Makoto Yoshida received the M.S. degree in Electrical Engineering from Kyushu University in 1999. In 2003, he joined the Fukuoka Industry, Science and Technology Foundation and worked on research and development of system LSI architecture. He is currently a researcher in the Institute of Systems, Information Technologies and Nanotechnologies.

Takayuki Kando was born in Aichi, Japan in 1967. He received his M.E. degree from Nagoya University in 1993. He had worked in FUJITSU LIMITED since 1993 to 1998. In there, He had engaged in research on computer algebra systems. He has worked at Quoliarc Technology Solutions Ltd. since 2008. His current research interests are program optimization, generative programming, and online service platform for development tools. He is a member of JSIAM, JSSAC, JSSST and IPSJ.

Yuichiro Ajima is a system architect in the Next-Generation Technical Computing unit at Fujitsu Limited. His research interests are in technical computing system architectures. He received a PhD in information engineering from the University of Tokyo. Ajima is a member of the Information Processing Society of Japan (IPSJ). Contact him at aji@jp.fujitsu.com.

Ikuo Miyoshi is a manager in the Next-Generation Technical Computing unit at Fujitsu Limited. His current research interests are in performance evaluation of HPC systems. He is a member of the Information Processing Society of Japan (IPSJ).

Toshiyuki Shimizu is a director in the Next-Generation Technical Computing unit at Fujitsu Limited. His research interests include high-performance computer system architectures, particularly interconnect architectures for highly scalable systems. Shimizu is a member of IPSJ and the Institute of Electronics, Information, and Communication Engineers. Contact him at t.shimizu@jp.fujitsu.com.

Yuji Oinaga is a Senior Vice President in the Next-Generation Technical Computing unit at Fujitsu Limited. He has been working for the development of mainframe and high performance computer systems.

Hisashige Ando received BS and MS from Tokyo Institute of Technology in 1968 and 1970 respectively. He joined Fujitsu in 1970 and worked for the development of multiple generations of high end computers. He later received Ph.D from Tokyo Institute of Technology in 2006. He retired from Fujitsu and currently is a technical writer specializing microprocessors and super computers. Also, he wrote three computer microarchitecture books and is teaching in multiple universities as a part time lecturer.

Yuichi Inadomi received the Ph.D. degree in science from University of Tsukuba, Japan in 1999. He joined the Department of Chemistry, University of Tsukuba in 1999, where he was a technical official. He is currently a post-doctoral fellow in the Research Institute for Information Technology, Kyushu University. His research interests include quantum chemistry and parallel programming. He is a member of the Chemical Society of Japan, Japan Society of Molecular Science, and the Information Processing Society of Japan.

Koji Inoue was born in Fukuoka, Japan in 1971. He received the B.E. and M.E. degrees in computer science from Kyushu Institute of Technology, Japan in 1994 and 1996, respectively. He received the Ph.D. degree in Department of Computer Science and Communication Engineering, Graduate School of Information Science and Electrical Engineering, Kyushu University, Japan in 2001. In 1999, he joined Halo LSI Design & Technology, Inc., NY, as a circuit designer. He is currently an associate professor of the Department of Advanced Information Technology, Kyushu University. His research interests power-aware computing, high-performance computing, dependable processor architecture, secure computer systems, 3D microprocessor architectures, and multi/many-core architectures. He is a member of the ACM, the IEEE, the IEEE Computer Society, and the IPSJ (Information Processing Society of Japan).

Mutsumi Aoyagi received his Ph.D degree from University of Nagoya in 1987. He started as a postdoc at Theoretical and Computational Chemistry group of Argonne National Lab. in 1988, after that, becoming Associate Prof. of Institute for Molecular Science in 1993 and finally Prof. at Research Institute for Informatics of Kyushu Univ. in 2002. His personal research area includes computational chemistry, large scale high performance computing, especially in regards to parallel computations and performance evaluations. He is member of IPSJ(Japan).

Kazuaki Murakami was born in Kumamoto, Japan in 1960. He received the B.E., M.E., and Ph.D. degrees in computer science and engineering from Kyoto University, Japan in 1982, 1984, and 1994, respectively. From 1984 to 1987, he worked for the Fujitsu Limited, where he was a Computer Architect of the mainframe computers. In 1987, he joined the Department of Information Systems of Kyushu University, Japan. He is currently a Professor of the Department of Advanced Information Technology, and also the Vice President of the Institute of Systems, Information Technologies and Nanotechnologies (ISIT). He is a member of the ACM, the IEEE, the IEEE Computer Society, the IPSJ (Information Processing Society of Japan), and the JSIAM (Japan Society for Industrial and Applied Mathematics).