NSIM-ACE: An Interconnection Network Simulator for Evaluating Remote Direct Memory Access

Ryutaro Susukita[1], Yoshiyuki Morie[1], Takeshi Nanri[1] and Hidetomo Shibamura[2]
[1]*Research Institute for Information Technology, Kyushu University,*
6-10-1 Hakozaki, Higashi-ku, Fukuoka 812-8581, Japan
[2]*Institute of Systems, Information Technologies and Nanotechnologies, Fukuoka SRP Center Building 7F,*
2-1-22 Momochihama, Sawara-ku, Fukuoka 814-0001, Japan

Keywords: Network Simulation, Interconnection Network, RDMA, Performance Evaluation, Parallel Processing, Discrete Event Simulation.

Abstract: Network simulation is an important technique for designing interconnection networks and communication libraries. Also network simulations are useful for the analysis of internal communication behavior in parallel applications. This paper introduces a new interconnection network simulator NSIM-ACE. This simulator enables us to evaluate RDMA directly while existing simulators do not have such capability. NSIM-ACE also provides a similar user-interface to RDMA-based parallel programs for easy use. The experimental evaluation indicates that the simulation accuracy is sufficient to compare performance of some RDMA-based algorithms and the simulator is capable of predicting performance scalability for non-extinct networks.

1 INTRODUCTION

Modern high performance parallel computers consist of a large number of computing nodes connected via an interconnection network. Applications running on such systems perform computation in parallel by communicating data between nodes. Designing high speed interconnection networks and communication libraries have an important role for running parallel applications efficiently. However, it is not an easy task to predict performance of interconnection networks and communication libraries at the design stage. Particularly, if a large number of nodes communicate a large amount of data simultaneously, communication performance degrades from the theoretical value due to communication contention. Therefore, simple mathematical models of parameters including the minimum communication latency between nodes and the bandwidth predict inaccurate communication performance. Analyzing communication behaviors inside real machines is also an issue for efficient parallel applications. It may be difficult because of the same reason. As an attempt to solve these problems, many interconnection network simulators were developed so far.

NSIM is an interconnection simulator developed for evaluating extreme-scale systems. Users can configure detail of the target network. This simulator focuses on simulation speed. The simulation model is simplified not at the great expense of simulation accuracy. A feature of NSIM is accepting programs compatible with Message Passing Interface (MPI) as input communication patterns. NSIM simulates the target network by means of pseudo execution of such a program. This feature provides with a user-friendly simulation environment. NSIM is implemented to be a parallel simulator on distributed memory systems.

Adiga et al. (2005) developed a dedicated simulator for predicting performance of three dimensional torus network of IBM BlueGene/L. The simulator inputs are traces of pseudo application codes. The developers extended an IBM tracer for generating the traces. This is a parallel simulator on shared memory systems. Simulations of up to a 64K-node network were reported.

BigNetSim (Choudhury et al., 2005) is a simulator supporting various interconnection networks. Users can flexibly configure network parameters including topology, size and communication latency. BigNetSim has two

Susukita, R., Morie, Y., Nanri, T. and Shibamura, H.
NSIM-ACE: An Interconnection Network Simulator for Evaluating Remote Direct Memory Access.
DOI: 10.5220/0005978802540261
In *Proceedings of the 6th International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH 2016)*, pages 254-261
ISBN: 978-989-758-199-1

execution modes. In one mode, it simulates according to communication patterns artificially generated inside the simulator. In the other mode, BigNetSim simulates according to traces generated by BigSim (Zheng et al., 2004), a simulator for extreme-scale parallel computers.

FSIN (Ridruejo and Alonso, 2005) is an interconnection network simulator included in a simulation framework called INSEE. FSIN supports a variety of router models and network topologies. This simulator is based on relatively simple network models where every network event spends equal time. FSIN accepts two types of simulation inputs, artificially generated communication patterns and traces of MPI applications. FSIN has a unique feature that it can feedback the simulation results to original trances.

SimGrid (Casanova et al., 2014) is a simulator for a wide area of computer systems including processor, network, storage and grid computing. SimGrid simulates these computer resources in a unified model where if multiple tasks on the target system require a resource simultaniouly, the resource is shared by the tasks in an optimization rule. This simulator accepts two types of simulation inputs, programs written in an original format and unmodified MPI programs.

Meanwhile, recent network interface cards (NICs) used in high performance parallel computers have the function of Remote Direct Memory Access (RDMA). RDMA has the following advantages:

- Low communication latency: RDMA directly transfer data from memory on the send node to memory on the receive node.
- Efficient parallelization of communication and computation: RDMA transfers data independently of node processors.
- Minimum consumption of memory: RDMA need no communication buffer.

Since MPI is the de fact standard of parallel programming, the RDMA-based programming model is not the main stream. However, the advantages above give RDMA a potential to become an alternative in coming high performance parallel computers.

Communication libraries that adopt the RDMA-based model include ARMCI (Nieplocha and Carpenter, 1999) and GASNet (Bonachea, 2002). These libraries have put/get operations in the RDMA-base model. MPI also defines put/get operations in addition to the message passing model. A recent example of communication libraries that support the RDMA-based model is the basic layer of ACP library (Sumimoto et al., 2016). It aims at a primitive communication library for parallel programing.

Unlike communications that require operations in both the send and receive nodes, RDMA starts the communication by either the send node or the receive node. On the other hand, RDMA often needs extra processing such as preparation for the communication and the confirmation of a write operation on the receive node. We need a different design of communication libraries and a different style of parallel programing.

However, existing interconnection network simulators do not provide the function of handling RDMA directly nor user-friendly interface to the RDMA-based programs. In order to break this limitation, we implemented the NSIM-ACE interconnection network simulator by extending NSIM. In this paper, we present this new simulator.

2 NSIM OVERVIEW

2.1 Network Model

NSIM supports mesh/torus networks up to six dimensions and fat tree networks. NSIM assumes that each node connected by the target network consists of one processor and one or multiple NICs. The target network is modeled as a combination of routers, router-to-router links and router-to-NIC links. The router model assumes static dimensional routing, virtual cut-through and a pipelined router. Data transfer on each link is simulated basically at a packet level. However, NSIM employs a simulation technique that gives the same accuracy as flit level simulations. For saving memory usage, actual data are not transferred but only information on data size is.

2.2 Inputs and Outputs

For communication patterns, NSIM accepts an MPI-like program in which the prefix MPI_ of MPI functions are replaced with MGEN_. This program is called an MGEN program. In the MGEN program, computational parts except for MPI functions are replaced with MGEN_Comp (t) functions, where t is a predicted computation time. This function enables us to simulate communication taking into account the difference of the start times between processes. It is also used for simulations including computation times. After a simulation, NSIM outputs a predicted execution time of the input MGEN program. Detailed statistics including the effective bandwidth and the effective usage of each link are also reported.

2.3 Modules

NSIM consists of the following five modules:

- DES (Discrete Event Simulation)
  A parallel discrete event simulator.
- MGEN (Message level event GENerator)
  MGEN performs pseudo execution of the input MGEN program and generates Message Level Events (MLEs). An MLE corresponds to a message in MPI.
- PGEN (Packet level event GENerator)
  PGEN calls MGEN for generating MLEs and generates Packet Level Events (PLEs) from the MLEs. A PLE is an event of DES.
- SIM (SIMulation control)
  SIM calls PGEN for generating PLEs and enqueues them to the event queue of DES. SIM also proceeds the simulation time of DES.
- EP (Event Processing)
  EP processes packet transfer on the network. It is the event procedure of DES.

2.4 Simulation Flow

At the beginning of a simulation, SIM calls PGEN. PGEN internally calls MGEN. MGEN generates MLEs from the MGEN program. It does not perform actual communications. Then PGEN generates PLEs from the MLEs. Each of these PLEs is an event that injects a packet into the network from a send NIC. SIM enqueues these PLEs to the event queue of DES and proceeds the simulation time of DES so that events are processed in correct time order. EP processes PLEs so that the packet is injected into the network. Also EP generates a new PLE that transfers the packet to a next link, a next router or the receive NIC. These PLEs are processed in correct time order by DES. Finally the packet reaches the receive NIC. The simulation is completed when all packets reach receive NICs.

3 NSIM-ACE

3.1 Simulation Model

NSIM-ACE models two types of RDMA, i.e., put and get operations. We focus put operations. In a put operation, the send process transfers data from memory on the send node to memory on the receive node. First, data on memory are transferred to the send NIC using DMA. The send NIC injects the data into the network as packets. The packets are transferred to the receive NIC on the network. Packet transfer on the network is modeled as the same way as in NSIM. If a packet of put data reaches to the receive NIC, it is transferred to memory on the receive node using DMA. After all put data are transferred to memory, the receive NIC sends a control packet to the send NIC. The put operation is completed when the control packet reaches the send NIC.

3.2 Inputs and Outputs

NSIM-ACE accepts an MGEN program for the communication pattern.

For describing RDMA in MGEN programs, we added new MGEN functions to NSIM.

- MGEN_acp_handle MGEN_rdma_put (int dest_rank, int data_size, int tag)
  MGEN_rdma_put issues a put operation which transfers data of data_size bytes from the process that calls this function to the process of rank dest_rank. The tag is used for specifying a put operation in the receive process. This function is completed even if the put operation has not completed yet. This function returns a handle that corresponds to the put operation.
- void MGEN_rdma_poll (int tag)
  MGEN_rdma_poll waits until the put operation specified by tag completes data transfer to memory on the receive node.
- void MGEN_acp_complete (MGEN_acp_handle handle)
  MGEN_acp_complete waits for the completion of the put operation specified by handle.

The functions and the type those have prefix of MGEN_acp_ in the names are similar to functions and type in ACP library those have names without MGEN_.

We show a sample MGEN program below.

```
#include "mgen.h"
int MGEN_Main(int argc, char
**argv){
    int rank, size, ms=4, r, tag=0;
    MGEN_Comm com = MGEN_COMM_WORLD;
    MGEN_acp_handle handle;
    MGEN_Comm_rank(com, &rank);
    MGEN_Comm_size(com, &size);

    r = (rank + 1) % size;

    handle = MGEN_rdma_put(r, ms,
 tag);
    MGEN_rdma_poll(tag);
    MGEN_acp_complete(handle);
 }
```

In this program, each process puts data to a neighbor process.

The NSIM-ACE simulates the target network according to the communication pattern described by the input MGEN program. After the simulation, NSIM-ACE outputs a predicted execution time and the same kinds of detail statistics as NSIM outputs.

3.3 Implementation

We implementation the put operation in a similar way to a send operation in NSIM. If a put operation is issued in the MGEN program, MGEN generates an MLE that corresponds to the put data. One difference from NSIM is that no receive operation is explicitly issued in the MGEN program. Another difference is the ordering of send and receive operations. In NSIM, send and receive operations are issued in order in the MGEN program. In NSIM-ACE, the control packet of the put operation is sent when all put data are transferred to memory independently of the MGEN program, i.e., the order is determined by intermediate results of the simulation. We added the following new flow to NSIM. After SIM proceeds the simulation time of DES, it checks packets that reach each NIC. If all packets of the put data reach a NIC, SIM calls PGEN. PGEN internally calls MGEN. MGEN generates an MLE that corresponds to the control packet. From the MLE, PGEN generates a PLE that injects the control packet into the network. SIM enqueues it to the event queue. This modification was not straightforward because NSIM was designed on the assumption that all send and receive operations are described and they are issued in order in the MGEN program.

Another consideration is the choice of the time step. DES was parallelized using conservative algorithm with lookahead. We assumed that a PLE of time t_1 that injects a control packet is generated and then enqueued after the simulation time is proceeded from t to $t + \Delta t$. The PLE is processed after the simulation time is proceeded to $t + \Delta t$. However, if Δt is too large, $t_1 < t + \Delta t$. Since any event of time between t_1 and $t + \Delta t$ is processed when the simulation time is proceeded to $t + \Delta t$, the PLE is processed in a wrong time order. We set Δt to the minimum latency of router-to-NIC links. The injection PLE is generated when DES processes a PLE of time t_2 that transfers the last packet of the put data from a router to a NIC between t and $t + \Delta t$. Since $t_2 > t$ and the latency from the router to the NIC is equal to or larger than Δt, $t_1 \geq t_2 + \Delta t > t + \Delta t$. This guarantees that the injection PLE is processed in correct time order. A similar discussion is applied to get operations.

We did not change EP, which determines the network model. The network is simulated in the same way as NSIM.

4 EXPERIMENTAL EVALUATION

In order to evaluate the simulation accuracy of NSIM-ACE, we compared simulation results and measurements on a real machine in three experiments. In addition, we predicted performance scalability beyond the number of processes in real measurements using NSIM-ACE.

4.1 Random Ring

The random ring traffic is one of High Performance Computing Challenge benchmark suite (Luszczek, 2006). Processes of the benchmark compose a ring in a random order. Each process sends 2MB data to the left and right neighbor processes in parallel and receives 2MB data from the left and right neighbor processes in parallel. The benchmark measures the bandwidth of the data transfer.

4.1.1 Experimental Environment

We ran the benchmark on Fujitsu PRIMERGY RX200 S7. Each node has one quad-core Intel Xeon processor E5-2609 (2.40 GHz). Sixteen nodes are connected by InfiniBand QDR switches. The throughput of each switch is 4GB/s per one direction. The port-to-port latency is 140 nanoseconds or below. The routing is destination-based. The original random ring traffic benchmark is written using MPI. We rewrote it using the put or get operation in the basic layer of ACP library. We used the InfiniBand implementation of ACP library. The InfiniBand implementation creates a communication thread per process. The communication thread are always running when the process is running. We ran only two processes per node for excluding the impact of the communication thread on the bandwidth. We ran two processes per node also in two other experiments. The original benchmark measures bandwidth in ten different random process orders. We measured only in one process order.

The simulation parameters are listed in Table 1. The DMA transfer speed was obtained by measuring the throughput in the random ring traffic benchmark of two processes on one node. We set the

communication library overhead to zero because it is so small compared to the one-hop latency of 2MB data transfer that it has little impact on simulation results. We set the other parameters according the specification of the real machine. In order to predict performance scalability above 16 nodes, we simulated with another parameter set where the parameters are the same except for that the number of nodes are 256.

Table 1: Configuration Parameters for NSIM-ACE.

Type	Parameter	Value
Router	Maximum theoretical communication speed of network	4.0 GB/s
	Switch throughput	4.0 GB/s
	Routing calculation time	4.0 ns
	Virtual channel allocation time	4.0 ns
	Switch allocation time	4.0 ns
	Switch latency	128 ns
	Cable latency	0.6 ns
Node	DMA transfer speed	2.8 GB/s
	Communication library overhead	0 ns
	Number of processes	One process / node

There are a few differences between the simulations and the real measurement. One is in the programs. The MGEN program was described by extracting communication parts of the original benchmark. NSIM-ACE assumes one process per node as NSIM. We described the MGEN program so that one process performs communications in parallel that correspond to two processes in the real machine. Another difference is intra-node communications. NSIM-ACE does not simulate intra-node communications. Instead we described MGEN_Comp (t), where t is the communication latency of 2MB data transfer in the node DMA. In addition, algorithms of arbitration and routing are different between the simulations and the real machine. These differences may cause simulation errors described below.

We performed simulations on HP ProLiant ML350e Gen8 v2 in all three experiments. It has two quad-core Intel Xeon processor E5-2407 v2 (2.4 GHz). NSIM-ACE required approximately 40 seconds for simulations when the number of process was 512. These times were measured in sequential executions including two other experiments.

4.1.2 Results and Discussions

The comparison of simulation and real machines are shown in Figure 1. Simulation results agree well with the real measurements at 4 and 8 processes. However, they show lower bandwidth than the real measurements at 16 and 32 processes. The difference may be caused by differences in arbitration and routing algorithms of the switches. In this experiment, all nodes on which processes are running are connected to the same switch if the number of processes is 4 or 8. In this case, the algorithm differences are unlikely to be seen. Otherwise, the nodes on which processes are running are connected to multiple switches. In this case, there are multiple communication routes between nodes. The simulation can be more likely to occur communication contention than the real measurements. As a result, it is possible to show lower bandwidth than the real measurements.

The 256-node simulations predict that the bandwidth gradually decreases above 32 processes. However, if we actually increase nodes and those connected to each switch for the real machine like the simulations, such a machine is expected to give higher bandwidth than the simulations because the real machine shows higher bandwidth than the simulations at 16 and 32 processes.

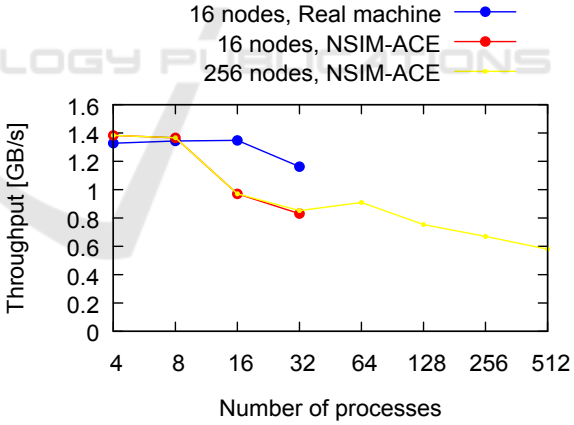

Figure 1: Random ring bandwidth of put operation.

4.2 Synchronization Barrier

Since a synchronization barrier communicates small data, simple mathematical modeling is effective for the performance prediction of the barrier itself. However, we need the barrier simulation if it is included in another communication pattern. For example, a barrier is used for waiting until multiple receive processes become ready for a put operation.

4.2.1 Ring Algorithm

Many algorithms are known for synchronization barrier of p processes. One of simple algorithms is ring. The complexity is $O(p)$. This algorithm can be implemented using put operations as follows. The ring algorithm performs a barrier in $p - 1$ steps. In each step, the process of rank i ($i \neq p - 1$) puts some data to some memory region of the process of rank $i + 1$. The process of rank $p - 1$ puts to the process of rank 0. After that, the process of rank i ($i \neq 0$) waits for receiving the data from the process of rank $i - 1$ by polling the memory region. The process of rank 0 waits for receiving from the process of rank $p - 1$. After $p - 1$ steps, the barrier is completed.

4.2.2 Recursive Doubling Algorithm

One of $O(\log p)$ algorithms is recursive doubling, which can be implemented using put operations as follows. If $p = 2^n$, the recursive doubling algorithm performs a barrier in n steps. In step i ($i = 1, 2, 3, ..., n$), 2^n processes are divided into groups of 2^i processes in the rank order. In each group, each process of smaller $2^i - 1$ ranks puts some data to some memory region of the process whose rank is the sender's rank + $2^i - 1$. Each process of larger $2^i - 1$ rank puts some data to some memory region of the process whose rank is the sender's rank $- 2^i - 1$. Then the two processes waits for receiving the data from each other by polling the memory region. After n steps, the barrier is completed. If $p = 2^n + r$ ($0 < r < 2^n$), this algorithm needs extra steps for r processes before step 1 and after step n.

4.2.3 Experimental Environment

We compared simulation results of barrier time to real measurements. We executed programs of both algorithms written using ACP library on the same machine as in the random ring experiment. In this experiment, we observed fluctuation in average barrier times larger than in usual measurements. We measured the minimum barrier time in 100 same barriers for obtaining the time required for the barrier at the least on this machine.

We used the same configuration parameters as in the random ring experiment except for the communication library overhead. In this experiment, the communication library overhead is not negligible compared to the one-hop latency of put data. We determined the communication library overhead by measuring the barrier time of two processes on one node. In this case, each of the two processes puts the data to each other. The barrier time corresponds to the communication library overhead excluding network latency. In this environment, half of the barrier time is closer because the NIC of the node performs two put operations sequentially. The determined value was 0.8 microseconds. For obtaining the communication library overhead fitting real measurements best, we also varied nearby 0.8 microseconds. The communication library overhead in configuration parameters corresponds to the node latency including the minimum latency of memory access and that of the node DMA transfer in addition to the communication library overhead. We call this parameter node latency, hereafter.

We described MGEN programs in a similar way to that in the random ring experiment.

When $p = 512$, NSIM-ACE needed 148 and 0.33 seconds for simulating the ring and recursive doubling algorithms, respectively.

4.2.4 Results and Discussions

The comparison of the simulation results and the real measurements is shown in Figure 2. If we set the node latency to the real measurement (0.8 μs), the simulation results are somewhat smaller than the real measurements in both algorithms. If we vary the node latency, the simulation results follow best the trend of real measurements in the case of 0.6 microseconds. The comparison of the two algorithms shows the recursive doubling algorithm is faster than ring in real measurement as expected. The simulation indicates the same result. Furthermore, we find that the recursive doubling algorithm is faster if $p = 2^n$ than otherwise in the real measurements. The simulations also give the same trend. In this experiment, the simulation accuracy of NSIM-ACE is sufficient to compare the two algorithms and reproduce the characteristics of the recursive doubling algorithm if we give an appropriate node latency. We conclude that NSIM-ACE is so useful for the performance prediction of barriers as simple mathematical modelling. Results of 256-node simulations followed as the algorithm complexities.

4.3 Particle Data Communication after Domain Decomposition

4.3.1 Communication Pattern

In parallel gravitational N-body simulations, particle data communications after domain decomposition are described more directly than point-to-point or collective communications of MPI (Susukita et al.,

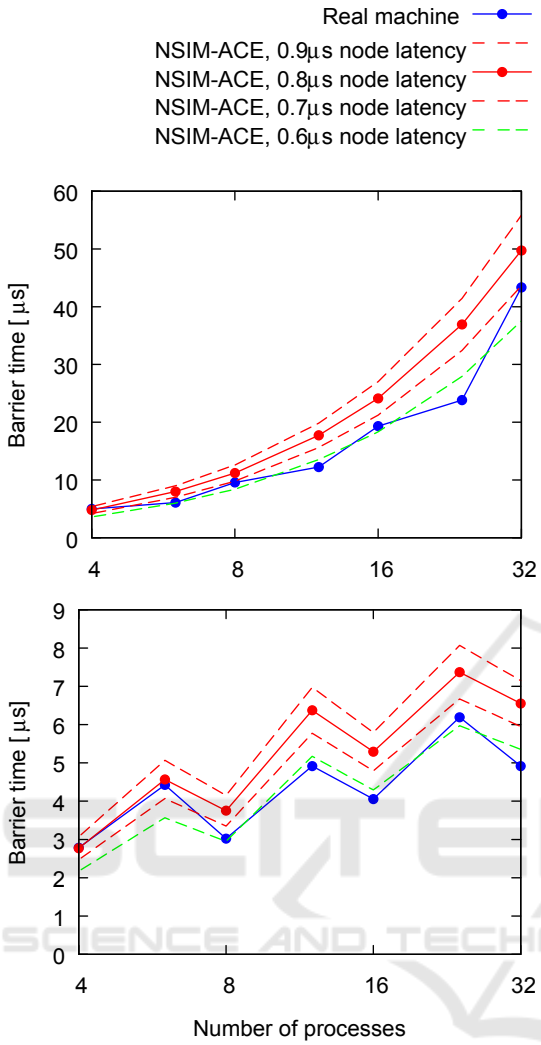

Figure 2: Barrier times of ring (top) and recursive doubling (bottom) algorithms.

2015). In this communication pattern, we need a synchronization barrier before the data packing or after that.

4.3.2 Experimental Environment

In a previous report (Susukita et al., 2015), we showed that simulation results of NSIM-ACE are in good agreement with real measurements in regard to this communication pattern. In addition, we found that the simulations distinguish performance change between two different synchronization points described later. In this paper, we simulated the communication pattern beyond the system size of the real machine.

The same environment was used as in the report. We executed with 64 and 128 processes on another real machine only for recording communication patterns. The largest put data decreased from 2.1 MB in the 32-process execution to 0.79 MB in the 128-process execution.

We described an MGEN program by extracting put operations of particle data. We also simulated a variant of the MGEN program in which we do not describe MGEN_Comp calls for intra-node communications, i.e., inter-node only variant.

NSIM-ACE required 0.65 and 0.83 seconds for the 128-process simulations when we performed the barrier before and after the packing, respectively.

4.3.3 Results and Discussions

The simulation results are shown in Figure 3. For 32 processes, differences between the simulation results and the real measurements are less than 10% both in the pack time and put operations. The simulations predict that the barrier before the packing provides better performance than the barrier after the packing up to 128 processes. This means that NSIM-ACE is able to propose a better synchronization point for a non-existent machine on which we cannot actually measure the performance.

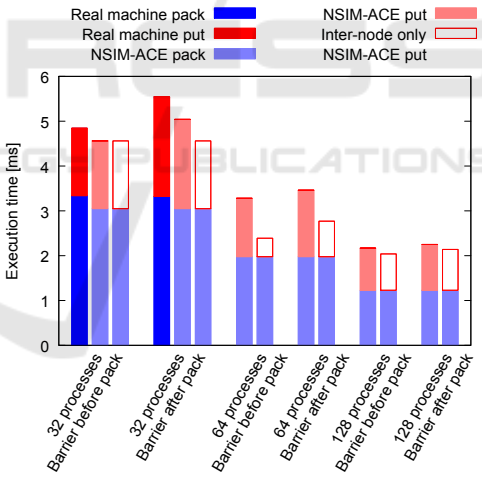

Figure 3: Execution time of particle data communication.

The simulations of the inter-node only variant indicate contributions of inter-node put operations. They suggest that both inter- and intra- node put operations are possible to make a great impact on the results. When the number of processes is 32, the latency of inter-node only simulations make little difference between the two synchronization points, while original simulations make a difference. We inferred that intra-node put operations make the difference. For example, intra-node put operations may cause communication contention if we

performed a barrier after the packing. By contrast, when the number of processes is 64, the latency of put operations is increased if we performed a barrier after the packing even in inter-node only simulation. In this case, inter-node put operations may cause communication contention.

5 CONCLUSIONS

In this paper, we introduced NSIM-ACE, a new interconnection network simulator for RDMA evaluation. We implemented it by extending NSIM simulator for large-scale interconnection networks. The NSIM-ACE has a user-friendly interface where the communication pattern is given in a similar way to RDMA-based parallel programs. We performed three experiments for evaluating the simulation accuracy and predicting performance scalability. The experiment on random ring bandwidth shows that the simulator produces bandwidth degradation due to communication contention. The experiment on synchronization barrier indicates that the simulation accuracy is sufficient to compare performance of RDMA-based algorithms and find algorithm characteristics. In addition, NSIM-ACE can predict the better algorithm for a communication pattern appearing in a particle simulation.

ACKNOWLEDGEMENTS

This work is supported by Core Research for Evolutional Science and Technology (CREST) Program of Japan Science and Technology Agency (JST), Research Area “Development of System Software Technologies for post-Peta Scale High Performance Computing”, Research Theme “Development of Scalable Communication Library with Technologies for Memory Saving and Runtime Optimization“. A part of computation was carried out using the computer facilities at Research Institute for Information Technology, Kyushu University.

REFERENCES

Adiga, N. R. et al., 2005. Blue Gene/L Torus Interconnection Network. *IBM Journal of Research and Development,* Vol.49, pp.265-276.

Bonachea, D., 2002. GASNet Specification, v1.1. *U.C. Berkeley Tech Report* (UCB/CSD-02-1207).

Casanova, H., Giersch. A., Legrand, A., Quinson, M. and Sutero, F., 2014. Versatile, Scalable, and Accurate Simulation of Distributed Applications and Platforms. *Journal of Parallel and Distributed Computing*, Vol.74, No.10, pp. 2899-2917.

Choudhury, N., Mehta, Y., Wilmarth, T. L., Bohm, E. J. and Kale, L.V., 2005. Scaling an Optimistic Parallel Simulation of Large-scale Interconnection Networks. *Proc. 37th Conference on Winter Simulation, Conference, WSC ’05*, pp.591-600.

Luszczek, P. R., Bailey, D. H., Dongarra, J. J., Kepner, J., Lucas, R. F., Rabenseifner, R. and Takahashi, D., 2006. The HPC Challenge (HPCC) Benchmark Suite. *Proc. 2006 ACM/IEEE Conference on Supercomputing, SC ’06.*

Miwa, H. et al., 2011. NSIM: An Interconnection Network Simulator for Extreme-Scale Parallel Computers, *IEICE Transactions on Information and Systems*, Vol.94, No.12, pp.2298-2308.

Nieplocha, J. and Carpenter, B., 1999. ARMCI: A Portable Remote Memory Copy Library for Distributed Array Libraries and Compiler Run-time Systems. *Proc. RTSPP of IPPS/SDP’99*.

Ridruejo, F. J. and Alonso, J. M., 2005. INSEE: An Interconnection Network Simulation and Evaluation Environment. *Proc. 11th Euro-Par Parallel Processing Conference 2005, Euro-Par’05*, pp.1014-1023.

Sumimoto, S., Ajima, Y., Saga, K., Nose, T., Shida, N. and Nanri, T. 2016. The Design of Advanced Communication to Reduce Memory Usage for Exa-scale Systems. *Proc. 12th International Meeting on High Performance Computing for Computational Science* (accepted).

Susukita, R., Morie, Y., Nanri, T. and Shibamura, H., 2015. Performance Evaluation of RDMA Communication Patterns by Means of Simulations, *Proc. 2015 Joint International Mechanical, Electronic and Information Technology Conference (JIMET 2015)*, pp.141-147.

Zheng, G., Kakulapati, G. and Kalé, L.V., 2004. BigSim: A Parallel Simulator for Performance Prediction of Extremely Large Parallel Machines. *Parallel and Distributed Processing Symposium, International,* Vol.1, p.78b.