Performance Evaluation of RDMA Communication Patterns by Means of Simulations

Ryutaro Susukita[1, a *], Yoshiyuki Morie[1,b], Takeshi Nanri[1,c] and Hidetomo Shibamura[2,d]

[1]Research Institute of 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

[a]susukita.ryutaro.491@m.kyushu-u.ac.jp, [b]morie.yoshiyuki.404@m.kyushu-u.ac.jp, [c]nanri@cc.kyushu-u.ac.jp, [d]shibamura@isit.or.jp

Keywords: parallel programming, RDMA, performance evaluation.

Abstract. Communication patterns that appear in RDMA-based parallel programming are different from those in the standard MPI programming. In this paper, we evaluated the performance of typical communication patterns in RDMA-based applications using simulations. The simulations predicted execution times with errors less than 10%. The simulation accuracy was sufficient to detect a performance change between two different synchronization points in an application. We also showed that these simulations were useful for analyzing the cause of a performance degradation in the communication pattern.

Introduction

The standard programming model for parallel computing on distributed memory systems is message passing at the present. In particular, Message Passing Interface (MPI) is popular for parallel applications. In this programming model, both of the send and receive processes issue some operations before the communication is started.

Other programming models are also proposed for distributed memory systems. One of those is based on Remote Direct Memory Access (RDMA). RDMA is the function of transferring data from memory on the send node to memory on the receive node without node processors. Recent Network Interface Cards (NICs) for interconnection networks of high performance parallel computers have this function. RDMA has the following advantages over send-receive type communications where both of the send and receive nodes need some actions when the send and receive processes issue the operations.

- Low communication latency. The send-receive type communication requires extra actions both on the send and receive nodes. RDMA needs actions only on either send or receive node. This reduces communication latency.
- Efficient parallelization of computation and communication. Computation is done by a node processor, while RDMA is controlled only by a NIC. If the node has multiple NICs, even multiple communications are efficiently parallelized.
- Minimum memory consumption. In general, the send-receive type communications requires communication buffers. RDMA transfers data directly from memory on the send node to memory on the receive node.

Several communication libraries including ARMCI [1], GASNet [2] support the RDMA-based model. MPI also defines the RDMA-based model, called put and get operations, in addition to the message passing model. ACP library is a recently released communication library that adopts the RDMA-based model. The RDMA communication patterns in this paper are written using the ACP library.

Meanwhile, the RDMA-based model requires different communication patterns from those in the message passing model. Let us consider a situation where one process sends data to multiple processes. Unlike the message passing model, each receive process issues no operation, so that they cannot control when the data are received in the program. Therefore, the programmer needs to guarantee that memory regions have been already prepared for receiving before the communication. Usually a barrier synchronization is employed before the communication for the guarantee. In the barrier synchronization, each process waits until all the other processes reach the synchronization point in the program. If the send process perform a barrier synchronization before the communication and each receive process performs it when the memory region has been prepared, the send process waits until the memory regions have been prepared.

Many authors evaluated performance of RDMA communication patterns on real machines [3,4,5,6,7,8]. However, for real measurements, the target systems are limited to exiting and available systems. When we design a communication library or a parallel application, it is desirable to evaluate the performance also on future systems and on systems of which we are not users. Simulations of communication patterns make it possible. In addition, simulations are useful for analyzing internal behaviors of real communications. For example, if we find a performance degradation in a communication, we can analyze the cause using simulations. In this paper, we present performance evaluation of RDMA-based communication patterns by means of simulations.

NSIM-ACE Simulator

This section outlines the NSIM-ACE simulator. If we can build an exact simulation model of the full system including node processors, node memory and the interconnection network, we are able to simulate communication patterns completely. In practice, such a model is too complex to build. A practical solution is simulating a simplified model. We used an interconnection network simulator NSIM-ACE for evaluating RDMA communication patterns. This simulator was implemented by adding RDMA functions to the NSIM simulator [9]. The implementation of NSIM-ACE is described elsewhere [10].

System Model. NSIM-ACE assumes that the target system consists of computing nodes, NICs and a network connecting the nodes. Each node has one processor and one or multiple NICs. The network is modeled as a combination of routers, router-to-router links and router-to-NIC links. NSIM-ACE supports several network topologies including commonly used fat tree networks. The network model in NSIM-ACE is the same as in NSIM. The detailed network model of NSIM is described in [9].

Simulation Model of RDMA. NSIM-ACE simulates the put and get operations of RDMA. Communication patterns discussed later use the put operation. In the put operation, the send process transfers data on memory of the send node to memory on the receive node.

First, the data is divided into packets and then each packet is transferred to the send NIC using Direct Memory Access (DMA) where the data are transferred only by the NIC without the node processor. The transfer time of each packet is expressed as T_{send} = (packet size) * T_{dma}, where T_{dma} is the given DMA transfer speed. Only one put operation is allowed at a time per NIC. If the second put operation is issued before the last DMA transfer of the first put operation is completed, DMA transfers of the second one are delayed. They start after the last DMA transfer of the first put operation is completed.

After the DMA transfer, the send NIC injects the packet into the network. The packet is transferred to the receive NIC through routers and links. The transfer time of the packet is expresses as $T_{network} = T_{latency}$ + (packet size) * T_{bw}, where $T_{latency}$ is the given minimum latency and T_{bw} is the given network bandwidth. Only one packet is allowed to be transferred per direction per link at a time. If a packet of another put operation reaches the router before the transfer of first packet is completed, i.e. communication contention occurs, the transfer of the second packet is delayed. It starts after the transfer of the first packet is completed. As a result, the network bandwidth is degraded from the theoretical value.

After the packet reaches the receive NIC, the packet is transferred to memory on the receive node using DMA.

The total transfer time of the packet is expressed as $T_{send} + T_{network}$. The values of T_{send} and $T_{network}$ may include delays.

When the DMA transfer of the last packet is completed, the receive NIC injects a special packet, called a control packet, into the network. The control packet is transferred to the send NIC through the network in the same way as normal packets. When the control packet reaches the send NIC, the put operation is completed.

The latency of the put operation is expressed as $T_{pp} + T_{end} - T_{start} + T_{control} + T_{pp}$, where T_{pp} is the given overhead of the communication library, T_{start} is the time when the first DMA starts on the send node, T_{end} is the time when the last DMA is completed on the receive node and $T_{control}$ is the transfer time of the control packet. The first and second T_{pp} terms express the overheads on the send and receive nodes, respectively. The value of $T_{control}$ includes a delay if any communication contention occurs.

Inputs and Outputs. An NSIM-ACE user prepares a communication pattern and configuration parameters for the simulation. The communication pattern is described as a pseudo RDMA-based program. In the program, RDMA is performed by calling the following functions.

1) MGEN_rdma_put (*dest*, *data_size*, *tag*): This function issues a put operation that transfers data of *data_size* bytes to the process *dest* with tag *tag*. This function returns a handle that corresponds to the put operation.

2) MGEN_rdma_poll (*tag*): This function waits until the data with tag *tag* are transferred to memory on the receive node. It is called by the receive process of the put operation.

3) MGEN_acp_complete (*handle*): This function waits until the put operation specified by handle *handle* is completed. It is called by the send process of the put operation.

We are also able to call functions similar to MPI functions such as the MGEN_Comm_rank function.

For computational time excluding communications, we call the following function.

4) MGEN_Comp (*t*): If we call this function, the subsequent put operation is issued after an interval of time *t*.

We show a sample communication pattern in Fig. 1. In this communication pattern, process 0 puts data of 4 bytes to process 1 with tag 0 after a computational time 100000 and then waits until the put operation is completed. Process 1 waits until put data with tag 0 are transferred.

The configuration parameters specify the network topology, the network size, T_{dma}, $T_{latency}$, T_{bw}, etc.

After the simulation is completed, NSIM-ACE reports the predicted execution time of the communication pattern with detailed statistics.

```
#include “mgen.h”
int MGEN_Main(int argc, char **argv){
   int rank, t = 100000, dest = 1, data_size = 4, tag = 0;
   MGEN_acp_handle_t handle;
   MGEN_Comm_rank(MGEN_COMM_WORLD,
&rank);

   if(rank == 0){
      MGEN_Comp(t);
      handle = MGEN_rdma_put(dest, data_size, tag);
      MGEN_acp_complete(handle);
   }else if(rank == 1)
       MGEN_rdma_poll(tag);
}
```

Figure 1 A sample communication pattern.

Experimental Validation

In order to validate usefulness of simulating RDMA communication patterns, we evaluated those on the real machine and using simulations and then compared the results. Unlike the RDMA communication pattern in [10], we incorporated the impact of intra-node communications.

Communication Pattern. We evaluated particle data communications accompanied by a domain decomposition in gravitational *N*-body simulations. It is a typical RDMA communication pattern which we cannot simply write using send-receive type communications. In the simulation, the simulation domain is decomposed into a number of subdomains. Each simulator process computes on particles in one subdomain. This corresponds to a mapping from positions to processes. The domain is decomposed so that workloads are balanced between processes as possible. With the proceeding of the simulation, some particles move into another subdomain. As a result, workload may be imbalanced between processes. If the load imbalance becomes too large, the domain is decomposed again in order to restore load balance. After the domain decomposition, some particles are computed on by a different process. Therefore, we need to transfer the particle data from the old process to the new process.

It is difficult to write this communication pattern using message passing functions of MPI. The old process determines to which process particle data are sent according to particle positions and the mapping from positions to processes. In MPI, the receive process needs to know from how many processes data are received. However, the new process does not have information on that. For determining from how many processes data are received, for example, we need to keep the mapping in the previous domain decomposition and count how many old processes computed on the particles in that mapping.

In the RDMA-based model, this communication pattern is written more directly. After a memory region is prepared for receiving, each process performs a barrier synchronization. Then the old process puts particle data to the new process and the new process does nothing. A similar discussion is found in [11].

Particle data sent to the same process are not always stored on a continuous region of memory. Because an ordinary put operation transfers data on a continuous region, we gather the particle data to a continuous region before the put operation. This operation is called pack.

In this communication pattern, we have two choices for the barrier point, before and after the pack operation. In either case, the particle data are transferred correctly. However, the performance of communication patterns may differ. We evaluated both cases.

Experimental Environment. We wrote a domain decomposition program using the ACP library. For particle data, we performed a 128^3 particle simulation using an *N*-body simulation code, GADGET-2[13]. We obtained two sets of particle data before and after a domain decomposition of the simulation. The data size is 48 bytes per particle in the domain decomposition program.

We executed the program on Fujitsu PRIMERGY RX200 S7. The machine consists of 16 nodes. Each node has one quad-core Intel Xeon processor E5-2609 (2.40 GHz) and one NIC. The nodes are connected via InfiniBand QDR switches. We ran two processes per node. We measured the execution time for the pack operations, the barrier synchronization and the particle data communications. In this environment, measured average communication latency showed more fluctuation than usual measurements. Instead, we measured the minimum time in 10 or 20 executions.

Simulation Inputs. We described a communication pattern program that corresponds to the pack operations, the barrier synchronization and the particle data

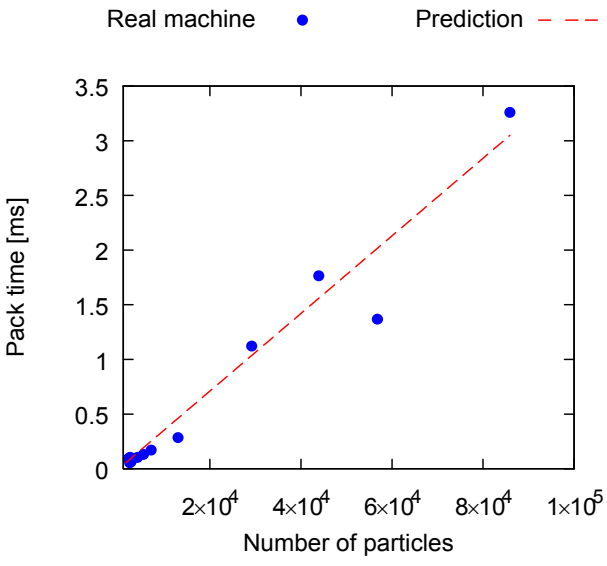

Figure 2 Predicion of pack time.

communications in the domain decomposition program.

For the pack operations, we called MGEN_Comp (*t*) in the communication pattern, where *t* is the predicted pack time. We assumed that *t* = (number of particles) * T_{pack}, where T_{pack} is the average pack time per particle. The value of T_{pack} was measured in two process run of the domain decomposition program on one node. The predicted pack times and real measurements are shown in Fig. 2.

The pattern of the particle data communications cannot be predicted before the simulation domain is actually decomposed. We recorded how much data and to which process each process puts in the domain decomposition program. The communication pattern program read the record and passed them to the MGEN_rdma_put function.

NSIM-ACE assumes one process per node. For simulating two processes per node, we described so that one process in the simulation parallelly performs communications which the corresponding two processes perform in the real machine.

When we run two processes per node, the particle data communications may include intra-node put operations. Intra-node means that the send and receive nodes are the same. In the ACP library, intra-node put operations are implemented using DMA. Because only one put operation is allowed per NIC at a time, the subsequent put operation should start after the intra-node put operation is completed. However, NSIM-ACE does not support intra-node put operations. Instead, we described MGEN_Comp (*t*) in the communication pattern program, where *t* is the predicted latency of the intra-node put operation. If we describe MGEN_Comp (*t*), the subsequent communication is issued after an interval of time *t* similarly to a computational time of *t*. We set *t* = (data size) * T_{dma} in the same way as in normal put operations. Conversely, if an intra-node put operation is issued before the last DMA transfer of the previous normal put operation is completed, the former should start after the latter is completed. Therefore, we also added MGEN_Comp (*t*) after each normal put operation.

Configuration parameters for NSIM-ACE are listed in Table 1. Router parameters were given according to the specification of the real machine. We determined the DMA transfer speed from real measurements. We used the random ring traffic in High Performance Computing Challenge benchmark suite [12] rewritten by using the ACP library. The DMA transfer speed was measured by running two processes that put 2 MB data to each other on one node. We set the overhead of the communication library to zero because it was negligibly small in the total communication latency.

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

Results and Discussions. We show the comparison results of real measurements and simulation results in Fig. 3. The times spent on the barrier synchronization are not drawn because those were very small. In both cases of different synchronization points, the total execution times were predicted with errors less than 10%. The times spent on the pack and put operations were also predicted within 10% errors, respectively. The execution times were smaller if we performs the barrier synchronization before the pack operation. If the synchronization barrier is performed after the pack

operation, put operations are issued almost at the same time in all processes. The reason why the performance was degraded may be the contention of these put operations.

For analyzing the reason further, we removed intra-node put operations in the communication pattern program. The simulation results (no intra-node in Fig. 3) show that the latency of put operations changed very little if the barrier synchronization was performed before the pack operation. However, the latency was reduced if the barrier synchronization was performed after the pack operation. These results suggest that the performance of put operations was degraded because of contention between intra-node put operations or between intra-node and normal put operations.

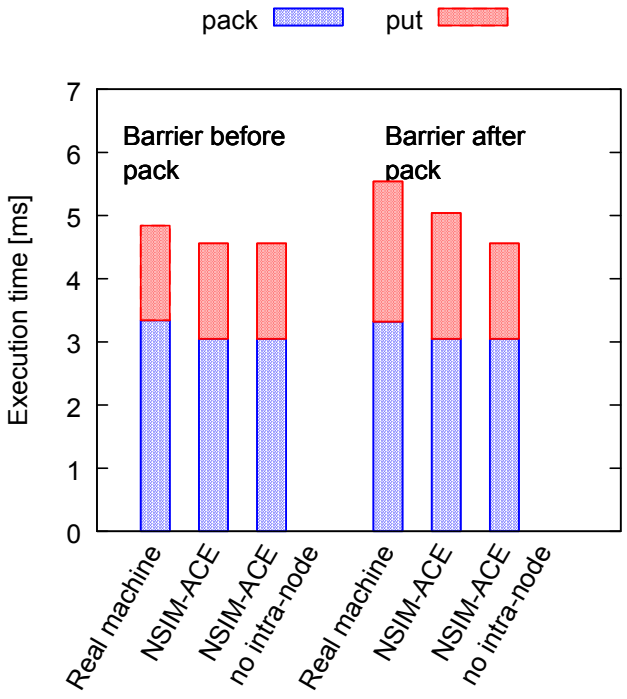

Figure 3 Comparison of real measurements and simulation results.

Conclusion

In this paper, we evaluated typical RDMA communication patterns that appear in gravitational *N*-body codes using simulations. The simulations gave execution times with errors less than 10%. The simulations were sufficiently accurate to detect a performance change due to different synchronization points. Furthermore, we showed that we are able to analyze the cause of performance degradation in the communication patterns by means of simulations.

Acknowledgment

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"

References

[1] J. Nieplocha and B. Carpenter, "ARMCI: A Portable Remote Memory Copy Library for Distributed Array Libraries and Compiler Run-time Systems," Proc. RTSPP of IPPS/SDP'99 (1999).

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

[3] W. Jiang, J. Liu, H-W. Jin, D. K. Panda, D. Buntinas, R. Thakur, and W. Gropp, "Efficient implementation of MPI-2 passive one-sided communication on InfiniBand clusters," Proc. 11th European PVM/MPI Users' Group Meeting, pp. 68-76 (2004).

[4] W. Jiang, J. Liu, H-W. Jin, D. K. Panda, W. Gropp, and R. Thakur, "High performance MPI-2 one-sided communication over InfiniBand," Proc. IEEE International Symposium on Cluster Computing and the Grid, 2004 (CCGRID 2004), pp. 531-538 (2004).

[5] M. Li, S. Potluri, K. Hamidouche, J. Jose, and D. K. Panda, “Efficient and truly passive MPI-3 RMA using InfiniBand atomics,” Proc. 20th European MPI Users’ Group Meeting, pp. 91-96 (2013).

[6] B. W. Barrett, G. M. Shipman, and A. Lumsdaine, “Analysis of implementation options for MPI-2 one-sided,” Proc. 14th European PVM/MPI Users’ Group Meeting, pp. 242-250 (2007).

[7] R. Gerstenberger, M. Besta, and T. Hoefler, “Enabling highly-scalable remote memory access programming with MPI-3 one sided,” Proc. International Conference on High Performance Computing, Networking, Storage and Analysis (SC ’13) (2013).

[8] J. Liu, J. Wu, and D. K. Panda, “High performance RDMA-based MPI implementation over InfiniBand,” Int. J. Parallel Prog., Vol. 32, No. 3, pp. 167-198 (2004).

[9] H. Miwa, et al., “NSIM: an interconnection network simulator for extreme-scale parallel computers,” IEICE Trans. Inf. & Syst., Vol.94, No.12, pp.2298-2308 (2011).

[10] R. Susukita, Y. Morie, T. Nanri, and H. Shibamura, unpublished.

[11] T. Hoefler, C. Siebert, and A. Lumsdaine, “Scalable communication protocols for dynamic sparse data exchange,” Proc. 2010 ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP ’10), pp.159-168 (2010).

[12] 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) (2006).

[13] V. Springel, “The cosmological simulation code GADGET-2,” Mon. Not. R. Astron. Soc., Vol. 364, pp.1105-1134 (2005).