MPI Library and Low-Level Communication on the K computer

● Naoyuki Shida ● Shinji Sumimoto ● Atsuya Uno

The key to raising application performance in a massively parallel system like the K computer is to increase the speed of communication between compute nodes. In the K computer, this inter-node communication is governed by the Message Passing Interface (MPI) communication library and low-level communication. This paper describes the implementation and performance of the MPI communication library, which exploits the new Tofu-interconnect architecture introduced in the K computer to enhance the performance of petascale applications, and low-level communication mechanism, which performs fine-grained control of the Tofu interconnect.

1. Introduction

The Message Passing Interface (MPI) communication library is a well-established standard for message passing in a parallel-processing system, but it would be no exaggeration to say that the quality of this communication library can have a significant effect on system performance.

This holds true for the K computer[note)] —to achieve a world-class level of performance in an 80 000-node, large-scale parallel-processing system, a variety of creative measures must be taken in using this library. In particular, it is essential that the communication time between nodes be minimized to raise performance. Thus, in addition to the MPI communication library as normally used, it is important that a low-level-communication mechanism for controlling the Tofu interconnect also be provided when implementing the MPI communication library.

This paper describes the MPI communication library and low-level-communication mechanism for the K computer.

2. Development targets and issues for implementing MPI communication library

Development targets and issues for implementing an MPI communication library are summarized below. These targets are not restricted to the K computer—they can also be applied to the design of supercomputers of the exaFLOPS class, which is 100 times the performance of the K computer.

1) MPI with high basic performance

Basic communication on a supercomputer includes point-to-point communication, collective communication, and MPI-IO, the performance of which can have a major impact on system performance. In particular, point-to-point communication performance, which contributes to collective communication performance and MPI-IO performance, requires an optimal communication system that can tap into the high performance of the hardware-based Tofu

note) "K computer" is the English name that RIKEN has been using for the supercomputer of this project since July 2010. "K" comes from the Japanese word "Kei," which means ten peta or 10 to the 16th power.

interconnect from the viewpoints of latency and bandwidth.

2) Optimal communication for a large-scale environment

System performance is generally proportional to the amount of memory used in communication, which means that there is a tradeoff between minimizing the amount of memory used in a system and achieving high performance. Additionally, memory usage increases in proportion to the number of processes with which a particular process must communicate, which means that running out of available memory is a high possibility in an 80 000-node-class system. How to go about minimizing memory usage given the above constraints is therefore an important issue. In short, there is a need for a communication system that can somehow minimize the amount of memory used.

3) User-friendly environment

The Tofu interconnect used in the K computer has a 6D mesh/torus topology.[1] Understanding its physical connections to develop programs is not an easy task for the general user. Studies must therefore be performed on how to go about bypassing faulty nodes in the operation of a large-scale system on a level of 80 000 nodes.

3. Overview of implementing MPI communication library

The policy that we adopted to implement an MPI communication library that can resolve the issues described in the previous section is to provide a low-level communication library tailored as much as possible to a standard application programming interface (API) based on an open-source MPI communication library and to achieve as far as possible the functions specific to the K computer through this low-level communication library. This would have the effect of minimizing changes to the standard MPI communication library.

The open-source MPI that we adopted for this purpose is Open MPI. We chose Open MPI first and foremost because it has a proven track record for the SPARC processors used in the K computer. Additionally, it supports InfiniBand, the main interconnect specification for PC clusters and thus simplifies the development process.

An MPI communication library for the K computer based on the above policy resolves the issues described above in the following ways.

1) MPI with high basic performance

First, to improve point-to-point communication performance, we provide a communication API centered about remote direct memory access (RDMA) communication that can make full use of the hardware characteristics of the Tofu interconnect at the level of a low-level communication library, and we use this API to implement an MPI communication library.

Second, to raise the performance of collective communication, we use the multiple network interfaces of the Tofu interconnect architecture on the basis of point-to-point communication centered about RDMA communication and adopt a collective-communication algorithm applicable to a 6D mesh/torus topology.

2) Optimal communication for a large-scale environment

We adopted two measures to achieve both high communication performance and low memory usage to the extent possible. The first measure was to minimize the amount of memory needed for a communication buffer by using communication centered about RDMA, and the second was to minimize memory usage by limiting to a fixed value the number of fellow processes with which a process can simultaneously communicate.

3) User-friendly environment

For the K computer, the 6D mesh/torus topology is presented as a virtual 3D torus to make it easier for users to handle the 6D topology. This is accomplished by combining the six axes of the 6D topology in such a way as to form a 3D configuration. As a result, it is easier to transplant applications previously developed for existing 3D-torus systems and there are more configurable 3D-torus shapes from which to choose. This 3D configuration results in a number of helpful features. For example, the system can be used as a 3D-torus system even if it is partitioned into multiple jobs, and applications can be executed without having to worry about faulty nodes in the system by appropriately setting the communication paths.

4. Implementation of Open MPI on the K computer

The structure of the MPI developed for implementation on the K computer is shown in **Figure 1**. Some changes have been made to the basic structure of Open MPI to support low-latency communication and collective communication based on RDMA. The following requirements were established for this implementation of Open MPI on the K computer.

1) Prompt upgrading to new versions

The MPI open-source library is constantly being updated to accommodate new functions, bug fixes, etc. In fact, there is a high possibility that the new functions to be added in MPI Version 3.0, which is now under study at the MPI Forum, will bring about major changes in the library. For this reason, much importance is being placed on an implementation whereby new versions of Open MPI can be accommodated by applying patches without having to modify the Open MPI structure if at all possible.

2) Focus on low-latency communication

Current implementations of Open MPI feature three communication-library layers (COLLs)—point-to-point messaging layer (PML), BTL management layer (BML), and byte transfer layer (BTL)—for achieving point-to-point communication. This means that at least three function calls must be made to perform such communication, which increases latency. To achieve the basic communication performance required of an MPI communication library for the

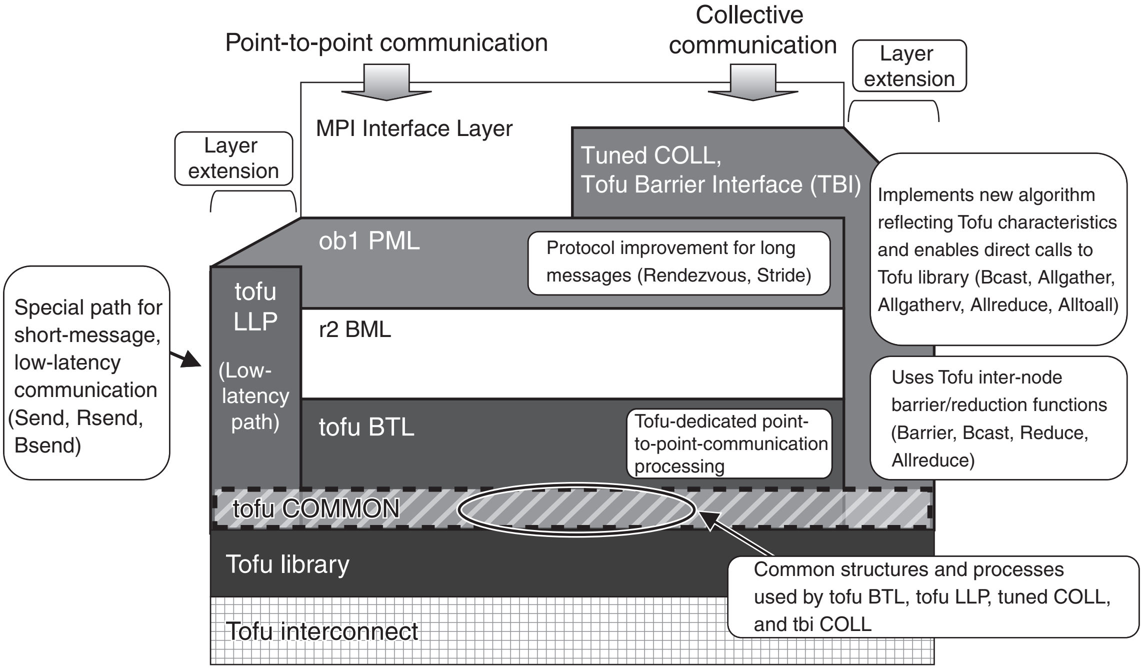

Point-to-point communication
Collective communication
Layer extension
MPI Interface Layer
Tuned COLL, Tofu Barrier Interface (TBI)
Layer extension
Implements new algorithm reflecting Tofu characteristics and enables direct calls to Tofu library (Bcast, Allgather, Allgatherv, Allreduce, Alltoall)
ob1 PML
Protocol improvement for long messages (Rendezvous, Stride)
Special path for short-message, low-latency communication (Send, Rsend, Bsend)
tofu LLP
(Low-latency path)
r2 BML
Uses Tofu inter-node barrier/reduction functions (Barrier, Bcast, Reduce, Allreduce)
tofu BTL
Tofu-dedicated point-to-point-communication processing
tofu COMMON
Tofu library
Common structures and processes used by tofu BTL, tofu LLP, tuned COLL, and tbi COLL
Tofu interconnect

Figure 1
Structure of MPI on the K computer.

K computer, a low latency path (LLP) has been added as a dedicated shortcut for point-to-point communication.

3) Collective communication based on RDMA

Part of the collective-communication algorithm uses a newly developed collective-communication framework instead of the standard Open MPI framework to enable

- Support of the Tofu hardware (Barrier, Bcast, Reduce, Allreduce)
- Support of RDMA-based collective communication
- Designation and control of multiple network interfaces.

In short, we have developed a collective-communication algorithm that enables multiple network interfaces to be used particularly for the frequently used communication operations of Bcast, Allgather, Allreduce, and Alltoall and that minimizes packet collisions on the network.

The amount of memory needed for communication purposes has been reduced by establishing two modes for MPI communication: high-speed and memory-saving. Communication begins in memory-saving mode, which keeps the size of the communication buffer small. However, if the number of times that communication is performed with another specific process exceeds a certain threshold, the mode is switched to high-speed, which uses a larger communication buffer. There is also a mechanism to prevent memory consumption from increasing by limiting the number of times that switching to high-speed mode can be performed.

5. Implementation of low-level communication on the K computer

The low-level-communication mechanism for the K computer taps the maximum performance of the Tofu-interconnect hardware through the use of a low-level communication library for the Tofu interconnect (Tofu library). This mechanism provides two types of functions.

1) Low-level communication

These are RDMA-based communication functions for tapping the full hardware performance of the Tofu interconnect.

2) Rank mapping

This function presents a 3D torus after avoiding faulty nodes on the basis of information received from the job interface.

6. Evaluation of basic communication performance

To assess the basic communication performance of the K computer, we first evaluated point-to-point communication performance for nearest-neighbor communication at both the Tofu library level and MPI library level. We then evaluated collective-communication performance by both hardware and software using the Allreduce and Barrier communication operations.

6.1 Point-to-point communication performance at Tofu library level

One-way communication latency in nearest-neighbor communication at the Tofu library level was 0.92 µs for an 8-byte message, and the maximum bandwidth was 4.76 GB/s for a 16-MB message. Given that hardware latency is 0.91 µs, it can be seen that communication processing latency at the Tofu library level was quite low at 0.01 µs.

Communication bandwidth performance for 1–4 Tofu network interfaces (TNIs) is shown in **Figure 2**. These results show that performance can be improved in a scalable manner up to 14.26 GB/s at 3TNI. The performance limitation at 4TNI of 15.03 GB/s is attributed to a bottleneck at the interface on the CPU side.

6.2 Point-to-point communication performance at MPI library level

The one-way latency at the MPI library level is shown in **Figure 3**, and communication bandwidth performance is shown in **Figure 4**.

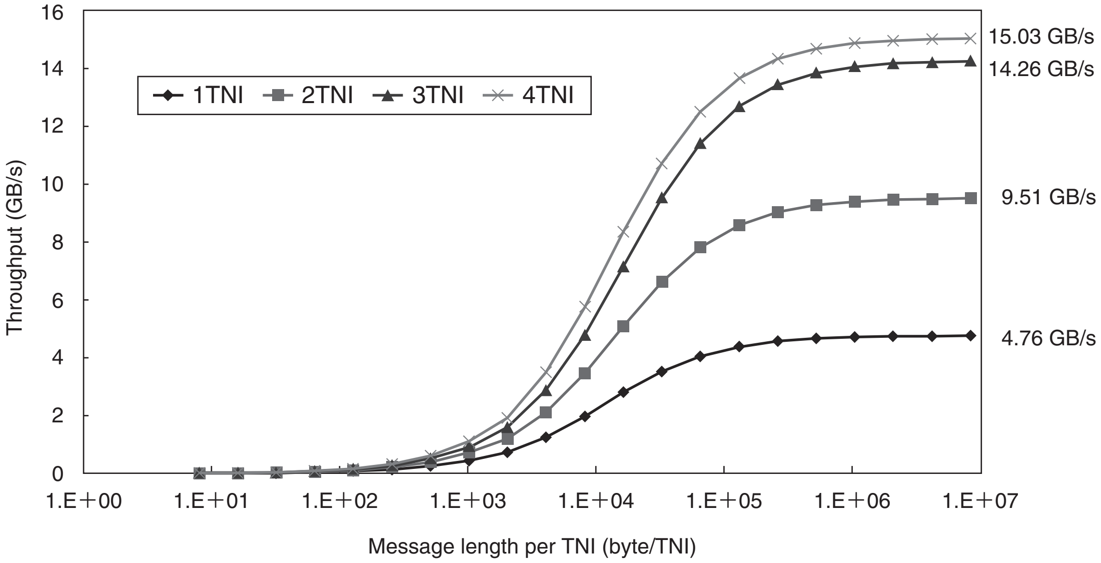


Figure 2
Communication bandwidth performance for 1–4 TNIs at Tofu library level.

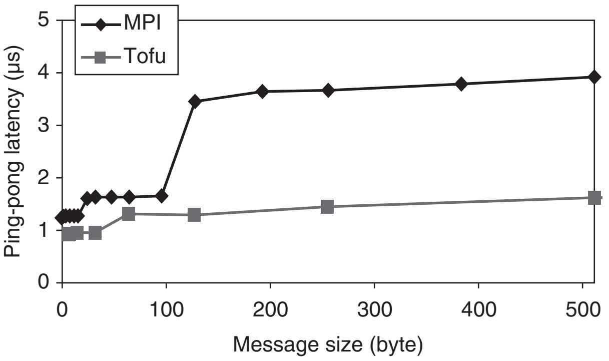


Figure 3
One-way latency at MPI library level.

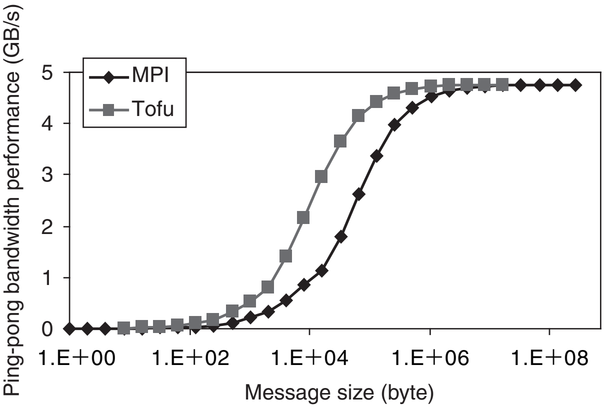


Figure 4
Bandwidth performance at MPI library level.

From the former figure, communication latency at the MPI library level was 1.27 μs for an 8-byte message in the case of nearest-neighbor communication, and from the latter figure, maximum communication bandwidth was 4.7 GB/s for a 16-MB message. It can be seen from these results that the hardware performance of the Tofu interconnect is being tapped to the maximum.

6.3 Collective communication performance at MPI library level

The collective communication performance for a hardware implementation using Tofu barrier interfaces (TBIs) at the MPI library level is shown in **Figure 5**. Specifically, this figure shows the results for the Allreduce and Barrier operations for up to 9216 nodes and compares these results with those of a comparable software implementation. Examining the results of the hardware implementation, it can be seen that there is practically no deterioration in

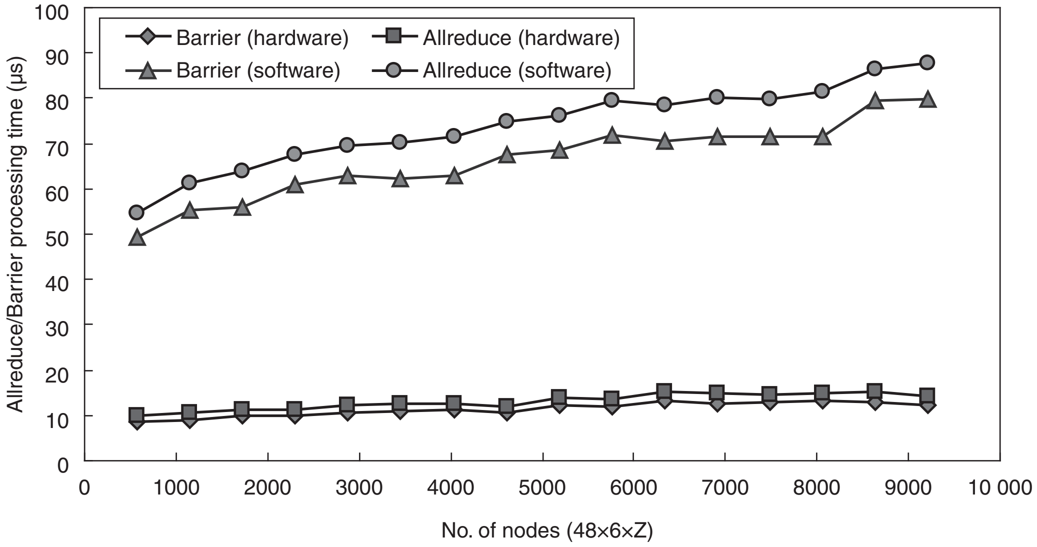
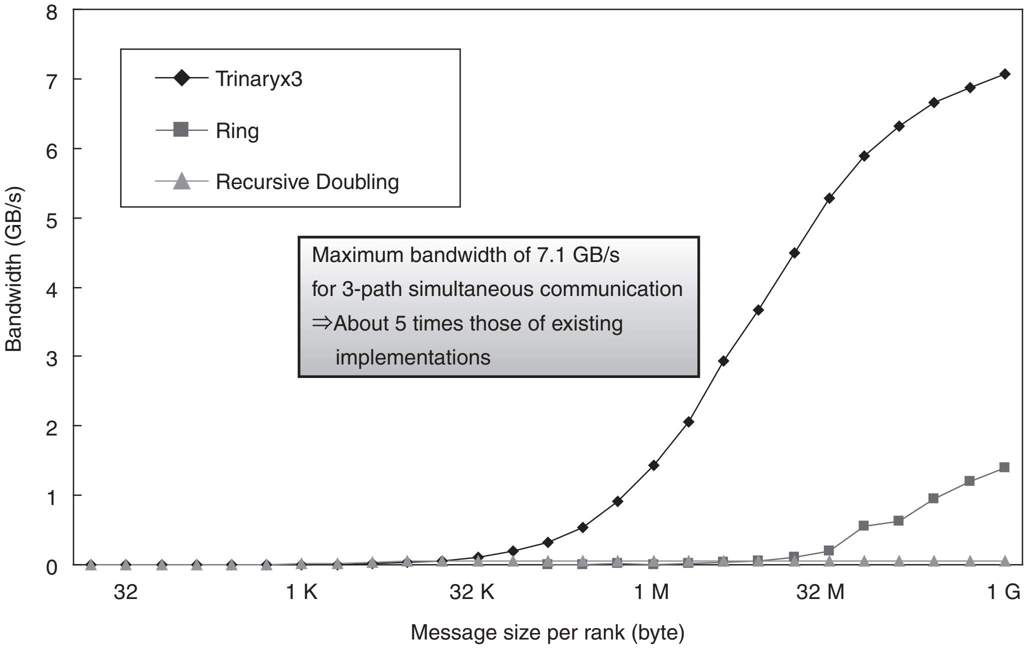


Figure 5
Hardware collective-communication performance at MPI library level.



Maximum bandwidth of 7.1 GB/s
for 3-path simultaneous communication
⇒About 5 times those of existing
implementations

Figure 6
Allreduce bandwidth performance at MPI library level.

performance for either operation even at 9216 nodes, indicating that performance was stable.

The bandwidth performance of the Allreduce operation at the MPI library level is shown in **Figure 6**. Specifically, the figure shows the results for a collective-communication algorithm (Trinaryx3) that uses multiple network interfaces to tap the full performance of the Tofu interconnect and that prevents the overlapping of communication paths. The Trinaryx3 Allreduce communication algorithm developed for the K computer uses three network interfaces and achieves a communication performance of 7.1 GB/s, which is about 5 times those of two existing collective-communication algorithms (Ring and Recursive Doubling), which are also shown for comparison.

7. Conclusion

This paper described the implementation and performance of the MPI and low-level-communication mechanism introduced in the K computer. Three development targets were established for this implementation: MPI with high basic performance, optimal communication for a large-scale environment, and user-friendly environment. To meet these targets, a variety of creative measures were taken, and the result was high communication performance and ease of use. Nevertheless, there is still much room for improvement on an ultra-large-scale machine like the K computer. In future research, We look to raise performance even further and to contribute to the Open MPI community.

References

1) Y. Ajima et al.: A 6D Mesh/Torus Interconnect for Exascale Computers. *IEEE Computer*, Vol. 42, No. 11, pp. 36–40 (2009).

Naoyuki Shida
Fujitsu Ltd.
Mr. Shida is engaged in the development of the MPI communication library.

Shinji Sumimoto
Fujitsu Ltd.
Mr. Sumimoto is engaged in overall technology development for HPC system software involving high-performance communication such as the MPI communication library and cluster file system.

Atsuya Uno
RIKEN
Dr. Uno is engaged in coordinating development of system software.