A Memory Saving Communication Method Using Remote Atomic Operations

Takahiro Kawashima
Fujitsu Limited
Japan
t-kawashima@jp.fujitsu.com

Masaaki Fushimi
Fujitsu Limited
Japan
fushimi.masaaki@jp.fujitsu.com

Takafumi Nose
Fujitsu Limited
Japan
nose.takafumi@jp.fujitsu.com

Shinji Sumimoto
Fujitsu Limited
Japan
sumimoto.shinji@jp.fujitsu.com

Naoyuki Shida
Fujitsu Limited
Japan
shidax@jp.fujitsu.com

ABSTRACT
The MPI library for the K computer introduced a memory saving protocol. However, the protocol still requires memory in proportion of the number of MPI processes and there can be shortage of memory if the number of processes reaches millions or tens of millions. Therefore, for the post-K computer development, we propose a new memory saving protocol using remote atomic operations in order to reduce memory usage for intermediate buffers. This protocol enables to allocate a shared receive buffer without contention of multiple sender remotely and asynchronously. Amount of memory used by an MPI library and communication performance when using our new protocol is shown in our poster.

CCS CONCEPTS
• Computing methodologies → Parallel computing methodologies;

KEYWORDS
MPI, remote atomic operation, point-to-point communication

1 INTRODUCTION
RIKEN and Fujitsu are now developing the post-K computer as a successor to the K computer. At the development of the MPI library for the K computer, memory usage of the MPI library was a big issue because existing general MPI libraries required memory in proportion of the number of MPI processes. In order to reduce the memory usage, we introduced two types of protocols which are high performance protocol and memory saving protocol based on Open MPI [2].

However, for the post-K computer development, it is very important that the MPI library will be able to run on at least ten times larger processes of the K computer. Therefore, further memory saving technology is required for the post-K computer.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

EuroMPI/USA 2017, September 2017, Chicago, IL, USA
© 2017 Copyright held by the owner/author(s).
With the shared receive buffer method, each process has a unique "Add" and "Or" operations as remote atomic operations, for example. This section describes the protocol of shared receive buffer method. The interconnect adopted to the post-K computer is also to inherit these features. Hereinafter, remote atomic operations are called "ARMW (Atomic Read-Modify-Write)".

3.2 Remote Atomic Operations

On distributed memory systems, remote atomic operations enable reading and updating remote memory atomically without a remote host CPU processing by combining the operations with RDMA Read/Write. In the Tofu interconnect 2 [1], there are "Fetch and Add" and "Or" operations as remote atomic operations, for example. The interconnect adopted to the post-K computer is also to inherit these features. Hereinafter, remote atomic operations are called "ARMW (Atomic Read-Modify-Write)".

**Fetch and Add:** Specified value is added to a value on the remote node memory atomically and the value on the remote side before that operation is written on the local node memory. Hereinafter, this operation is called "ARMW-add".

**Or:** This is "Bitwise OR" operation with specified value and a value on the remote node memory. The value on the remote side before that operation is written into the local node memory. Hereinafter, this operation is called "ARMW-or".

3.3 Protocol of Shared Receive Buffer Method

This section describes the protocol of shared receive buffer method. With the shared receive buffer method, each process has a unique intermediate buffer called "shared receive buffer" and the access to the buffer is controlled using two counter variables called producer counter and consumer counter.

**Shared Receive Buffer:** A unique intermediate ring buffer for each process. Used to receive control messages sent in rendezvous protocol or data in eager protocol.

**Producer Counter:** A counter that means the next write position in the shared receive buffer. Updated by communication peer processes.

**Consumer Counter:** A counter that means the position already read in the shared receive buffer. Updated by its owner process.

There are two benefits in this protocol. First, one intermediate buffer per process is enough regardless of the number of communication peers. Second, multiple senders can send data to the same receiver without CPU processing by the receiver basically. The protocol is described as follows and the figure 1 describes its overview.

1. The sender process increments the producer counter of the receiver process by the size in bytes of sending data using ARMW-add. The purpose of this ARMW is getting the position and reserving a region on the shared receive buffer of the receiver process for writing data.
2. The sender process performs an ARMW-or to the consumer counter of the receiver process in order to get the value of the counter. The value of the consumer counter is not changed because the operand of the ARMW-or is zero.
3. The sender process determines whether it is safe to write data based on the values of the producer/consumer counters obtained in (1) and (2). If it is determined not to be able to write data, the sender returns to (2). Otherwise, the sender sends data.
4. The receiver process confirms the data arrival by polling the shared receive buffer and copy the received data to the buffer specified by user.
5. The receiver process clears the region of the received data in (4) by zero and increments the consumer counter by the received data size.

4 SUMMARY AND FUTURE WORK

We presented the shared receive buffer method using ARMW as a memory saving communication protocol. This protocol enables to allocate a shared receive buffer without contention of multiple sender remotely and asynchronously. We are developing and debugging a prototype implementation. In our poster, we will describe its communication performance and memory usage.

ACKNOWLEDGMENTS

Thanks to RIKEN and project members of Fujitsu.

REFERENCES