A Bandwidth-saving Optimization for MPI Broadcast Collective Operation

Huan Zhou, Vladimir Marjanovic, Christoph Niethammer, José Gracia

HLRS, Uni Stuttgart, Germany

P2S2-2015 / Peking, China / 01.09.2015

Outline

Outline

What are MPI collective operations?

What is Message Passing Interface(MPI)?

- A portable parallel programming model for distributed-memory system
- Provides point-to-point, RMA and collective operations

What are MPI collective operations?

- Invoked by multiple processes/threads to send or receive data simultaneously
- Frequently used in MPI scientific applications
  - Use collective communications to synchronize or exchange data
- Types of collective operations
  - All-to-All (MPI_Allgather, MPI_Allscatter, MPI_Allreduce and MPI_Alltoall)
  - All-to-One (MPI_Gather and MPI_Reduce)
  - One-to-All (MPI_Bcast and MPI_Scatter)

Why is MPI_Bcast important?

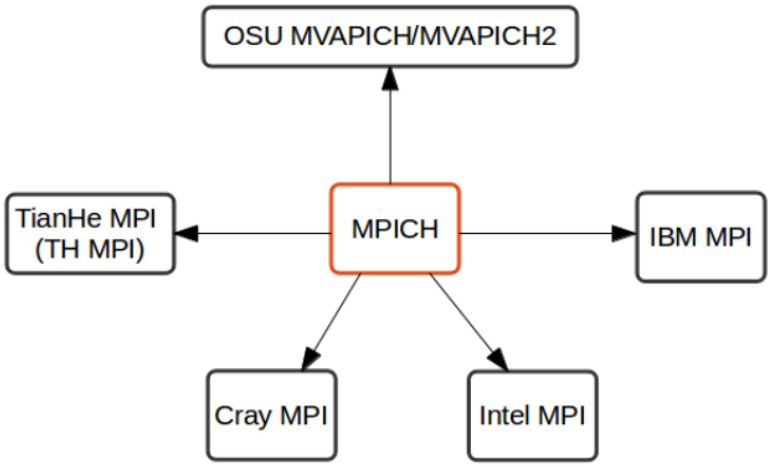
MPI_Bcast

- A typical One-to-All dissemination interface
  - The root process broadcasts a copy of the source data to all other processes
- Broadly used in scientific applications
- Profiling study shows its impact on application performance (LS_DYNA software performance)

Calls for optimization of MPI_Bcast!

Why is MPICH important?

- A portable, frequently-used and freely-available implementation of MPI.
- Implements the MPI-3 standard
- MPICH and its derivatives play a dominant role in the state-of-art Supercomputers

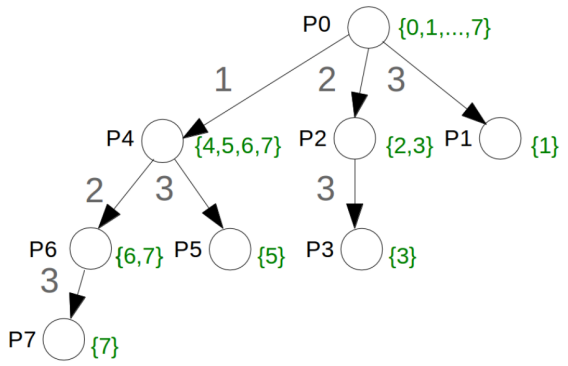


Outline

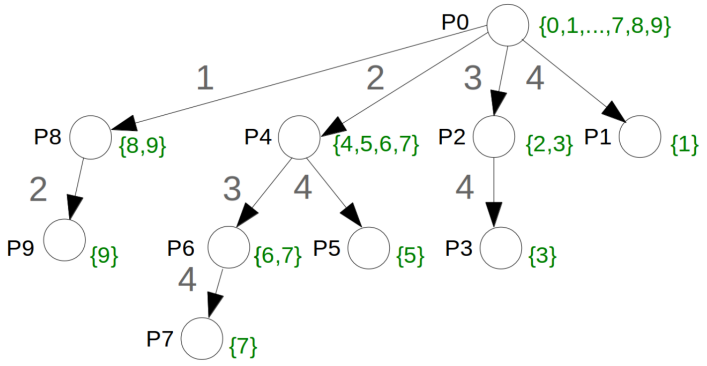
MPI_Bcast in MPICH3

- Multiple algorithms are used based on the message size and process counts
- A *scatter-ring-allgather* approach
  - Is adopted in case where long messages (*lmsg*) are transfered or in case where medium messages are transferred with non-power-of-two process counts(*mmsg-npof2*)
  - Consists of a binomial scatter and followed by a ring allgather operation
- MPI_Bcast_native is a user-level implementation of *scatter-ring-allgather* algorithm
  - Without multi-core awareness

The binomial scatter algorithm



8 processes (third-power-of-2)



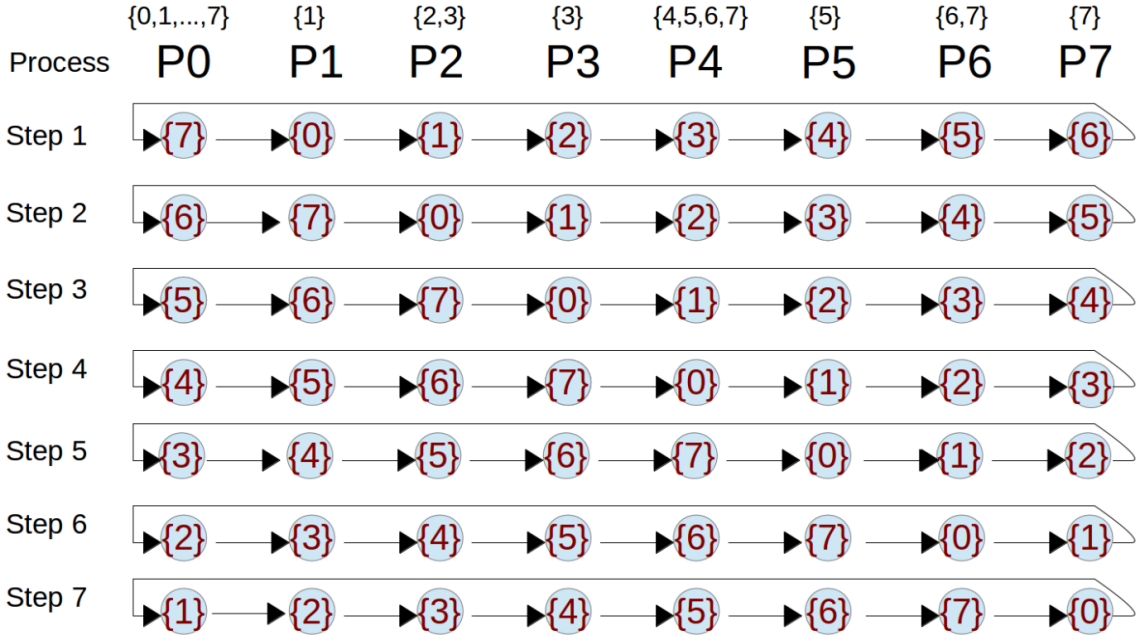
10 processes (non-power-of-2)

- Completed in $3 = log_2 8$ steps
- The root 0 divides the source data into 8 chunks, marked with 0,1,...,7, sequentially

- Completed in $4 = \lceil log_2 10 \rceil$ steps
- The root 0 divides the source data into 9 chunks, marked with 0,1,...,9, sequentially

Theoretically, process i is supposed to own data chunk i in the end
Practically, Non-leaf processes provide all data chunks for all their descendant

The native ring allgather algorithm



8 processes (enclosed ring)

- 7 steps and 56 data transmissions in total
- P0, P2, P4 and P6 repeatedly receive the data chunks that already existed in them
  - Bring redundant data transmissions
- This algorithm is not optimal

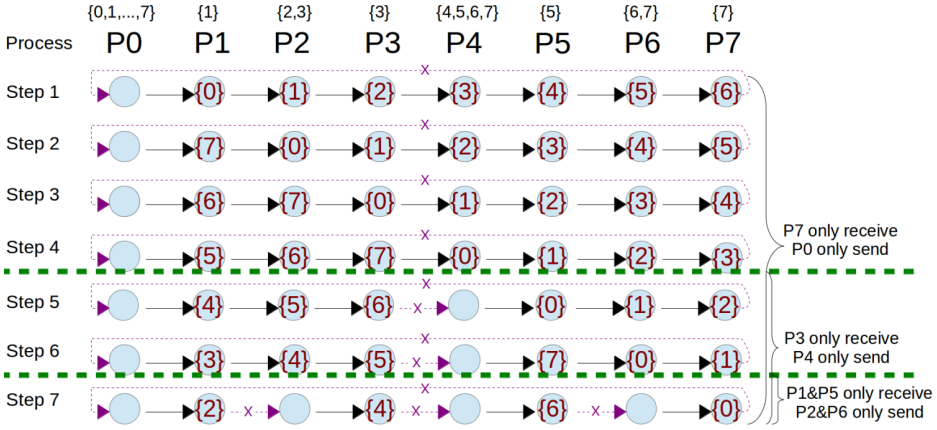
Motivation

- Especially, for *lmsg*, the usage of bandwidth is important
- The native ring allgather algorithm can be optimized
  - Avoid the redundant data transmissions
    - Each data transmission corresponds to a point-to-point operation
    - Save bandwidth use
    - Potentially bring reduction in communication time

Outline

The tuned design of the native scatter-ring-allgather algorithm I

- MPI_Bcast_opt is a user-level implementation of the tuned *scatter-ring-allgather* algorithm
  - Without the multi-core awareness
  - Leave the scatter algorithm unchanged and tune the native allgather algorithm
- The tuned allgather algorithm in the case of 8 processes

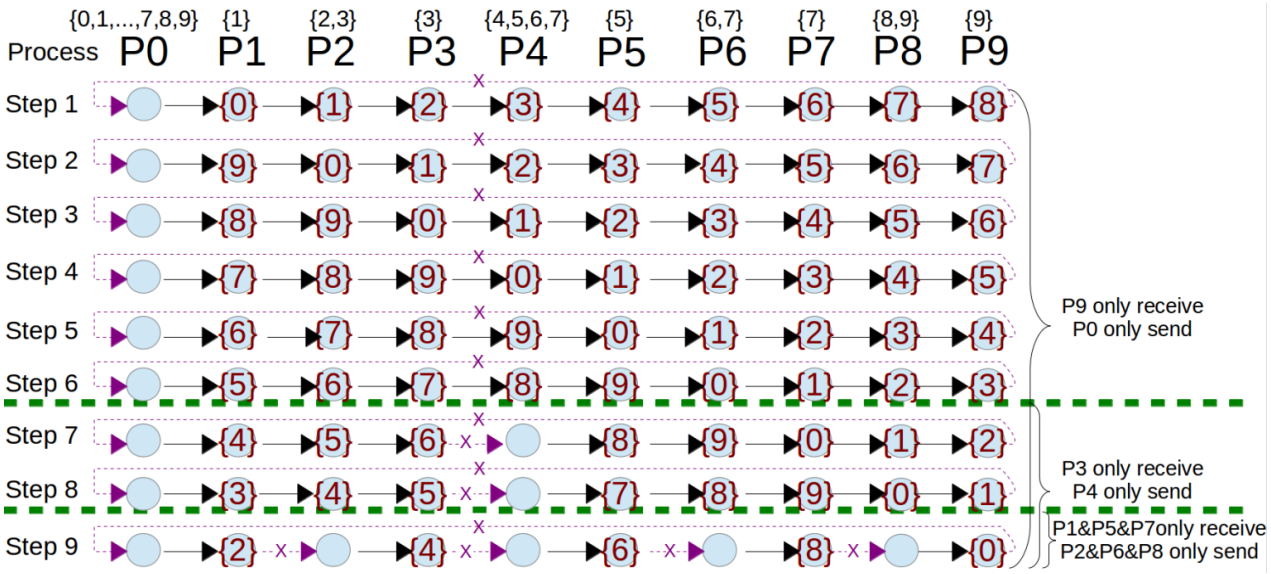


Non-enclosed ring for the tuned allgather algorithm

7 steps and 56 data transmissions in total, 12 data transmissions can be saved

The tuned design of the native scatter-ring-allgather algorithm II

- The tuned allgather algorithm in the case of 10 processes



9 steps and 75 data transmissions in total, 15 data transmissions can be saved

- The above two graphs show us that each process sends or receives message segments adaptively according to the chunks it has already owned

A brief pseudo-code for the tuned allgather algorithm

```
In: step, flag, comm_size
// Collect data chunks in (comm_size-1) steps at most
for i=1...comm_size-1 do
 // Each process uses step to judge if it has reached the point
 that indicates send-only OR recv-only
 if step ≤ comm_size-i then
   //The process sends and meantime receives data chunk
   to/from its successor/predecessor
   MPI_Sendrecv
 else
  // The process reaches the send-only point
   if flag = 1 then
     MPI_Recv
   // The process reaches the recv-only point
   else
     MPI_Send
   end if
 end if
end for
```

Advantages of the tuned ring allgather algorithm

The tuned ring allgather algorithm is analyzed in two communication levels – intra-node (within one node) and inter-node (across nodes)

Less intra-node data transmissions

- Reduces the cpu-interference
- Reduces the amount of memory(buffer)-allocation

Less inter-node data transmissions

- Besides less buffer allocation, it achieves reductions in network utilization

Outline

Experimental setup

- Evaluation platforms

Platform	Processor	Cores per node	Interconnect	MPI
Cray XC40 (Hornet)	Intel Haswell E5680v3 2.5GHz	24	Cray Aries Dragonfly topology	Cray MPI
NEC Cluster (Laki)	Intel Xeon X5560 2.8GHz	8	InfiniBand fabric topology	MPICH

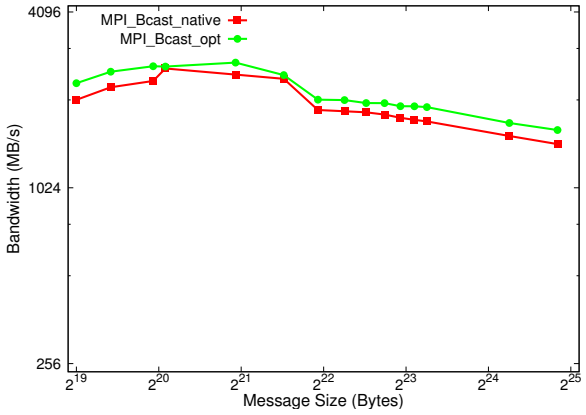
- Evaluation objects
  - MPI_Bcast_opt
  - MPI_Bcast_native

Only the results from Hornet are presented below since the results from Laki basically show the same performance trend

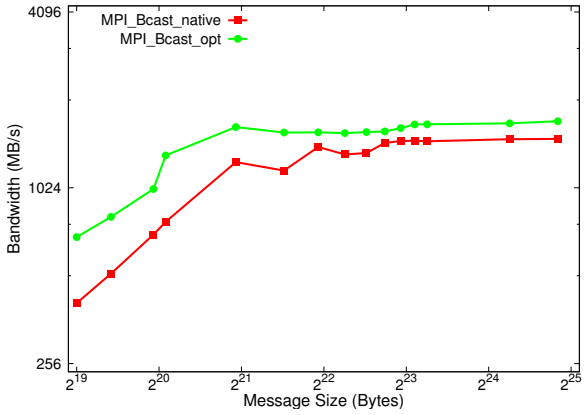
Experimental benchmarks

- The performance measurement metric: *Bandwidth*
  - Used to measure how fast the broadcast operations can be processed
  - Is simply the volume of broadcast messages finished divided by the complete time (measured in Megabytes per second (MB/s))
- Low-level benchmarks
  - Benchmark A: The *Bandwidth* of broadcast for a range of long messages ($\geqq$512KB and $<$32MB) with power-of-two processes
  - Benchmark B: The speedup of tuned broadcast over native broadcast for several medium messages ($\geqq$12KB and $<$512KB) and a long message (1MB) with non-power-of-two processes
  - Benchmark C: The *Bandwidth* of broadcast for a range of medium messages and long messages ($\geqq$12KB and $<$3MB) with 129 processes

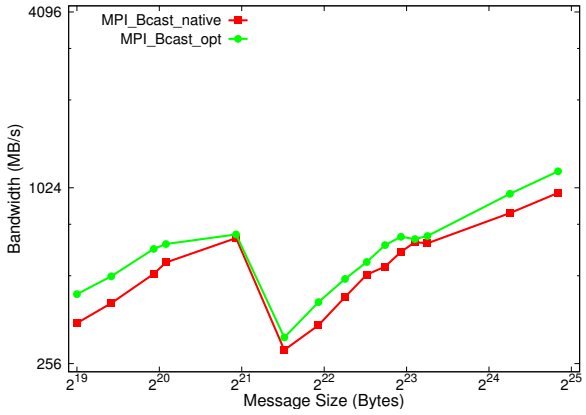
A: long messages($\geq$512KB and $<$32MB), power-of-two processes



np=16 (within one node)



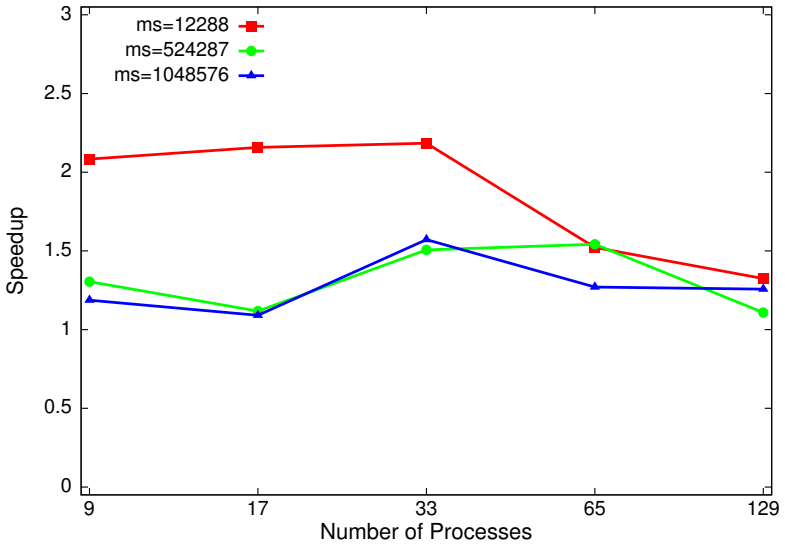
np=64 (across 3 nodes)



np=256 (across 12 nodes)

- The MPI_Bcast_opt performs consistently better than MPI_Bcast_native
- The drop in bandwidth performance of broadcast starts from around 3MB for 16 processes
  - limited memory bandwidth
- A sudden drop at around 3 MB for 64 and 256 processes
  - Cache effects

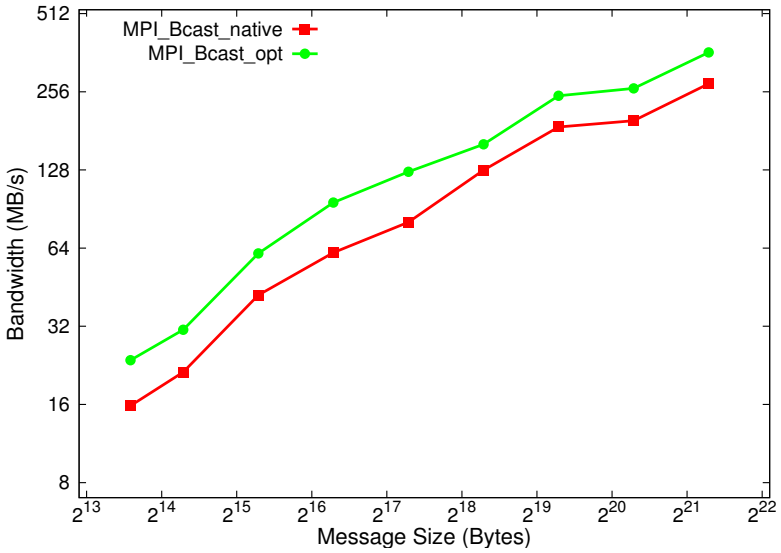
B: medium messages and long message, non-power-of-two processes



Throughput speedups of MPI_Bcast_opt over MPI_Bcast_native

- Two critical medium messages (take 12288 bytes(12KB), 524287 bytes(512KB-1) for example), one long message (take 1048576 bytes(1MB) for example)
  - MPI_Bcast_opt can perform more than two times better than MPI_Bcast_native for message size of 12KB
  - MPI_Bcast_opt performs consistently better MPI_Bcast_native with non-power-of-two processes

C: medium to long messages($\geqq$12KB and $<$3MB), fixed process counts



np=129

- MPI_Bcast_opt consistently performs better than MPI_Bcast_native
- The bandwidth increases steadily as the growth of message sizes
  - Sufficient memory resource

Conclusions

- Calling for the optimization of MPI_Bcast
  - Targets the long messages and medium messages with non-power-of-two processes
  - Brings in the tuned ring allgather algorithm
    - Reduces the amount of data transmission traffic
    - Eases the burden of network and host processing
  - Performs consistently better than the native one in terms of bandwidth

Thank You!

zhou@hlrs.de

MPICH Web Page

https://www.mpich.org/

Backup slides

Profiling study

- This profiling study shows the effect of MPI collective operations on LS-DYNA performance
- LS-DYNA software from Livermore Software Technology Corporation is a general purpose structural and fluid analysis simulation software package capable of simulating complex real world problems.
- LS-DYNA relies on the Message Passing Interface (MPI) for cluster or node-to-node communications
- The profiling results show that MPI_Allreduce and MPI_Bcast (Broadcast) consume most of the total MPI time and hence is critical to LS-DYNA performance

Why do we implement the user-level broadcast algorithm without multi-core awareness?

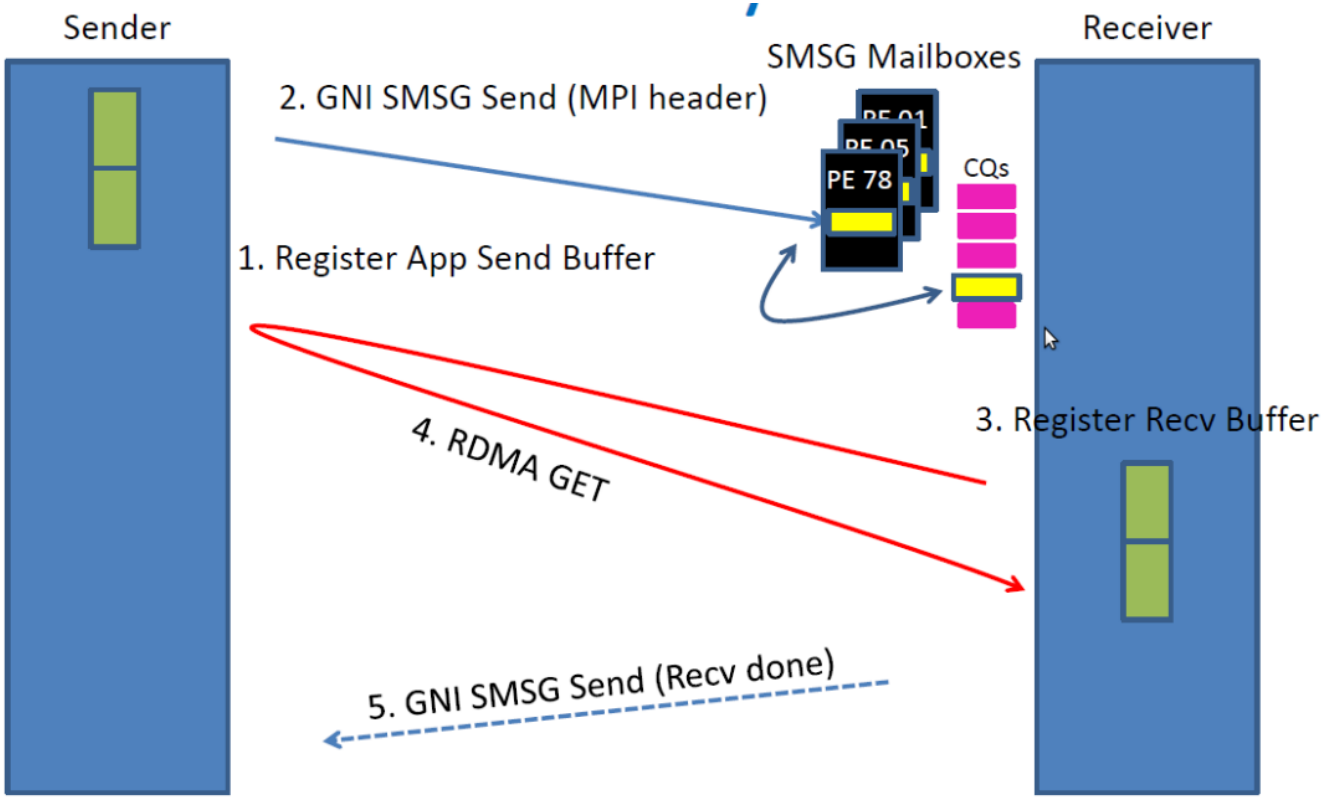
- The MPI_Bcast in MPICH is aware of the multi-cores.
  - The purpose of doing so is minimize the inter-node communication.
- The purpose of ignoring the multi-cores
  - For the user-level code simplicity
  - We also want to see the performance comparison between the native algorithm and the tuned one when there are a large number of inter-node data transmissions.

For the scatter-ring-allgather algorithm, the number of the saved data transmissions is the increasing function of the process counts

- I would better to generalize the relationship between the number of the saved data transmissions and the involved process counts, but it is kind of hard
- Assuming the original involved process counts is P, there will be P-1 steps in total, and with respect to the root, P-1 data transmissions can be saved. then when we involve a new process in the scatter-ring-allgather algorithm, the involved process counts are increased to P+1. In this case, there are in total P steps, during each step, the root will only send message and will not receive any message. With respect to the root, P data transmissions can be saved. Therefore, we can at least save one more data transmission.

Advantages

- Intra-node
  - First send the buffer information (for the long message)
  - Copy the data into and out of the shared memory
    - ★ involve the interference from cpu
    - ★ memory allocation
- Inter-node (adapted from the report from 'MPI on the Cray XC30')



Performance benefit

- Benchmark A
  - 16 processes: improved by 12% at best, report the peak bandwidth of up to 2748MB/s compared to 2623MB/s reported by the native one.
  - 64 processes: improved by 41% at best
  - 256 processes: improved by up to 20%
- Benchmark C
  - the bandwidth can be improved by 30% in the best case