Collective Communication Optimizations

SHARED MEMORY, OFFLOAD, AND NON-BLOCKING

MAMZI BAYATPOUR, CSE 5194, DR PANDA, MARCH, 1, 2016
Outline

- Multi-Leader-Based Allgather
- Non-blocking Broadcast with Collective Offload
- Functional Partitioning Approach to Design High-Performance
- Non-blocking Personalized Collectives
Designing Multi-Leader-Based Allgather Algorithms for Multi-Core Clusters

K. KANDALLA, H. SUBRAMONI, G. SANTHANARAMAN, M. KOOP AND D. K. PANDA
CAC '09, IN CONJUNCTION WITH IPDPS '09.
Overview

MPI collective operations take a significant portion of the communication time

- for an application Leader based design

Having large number of cores per node

- Need to have multi-core aware designs
  - to extract the best performance.
Allgather Communication

Each process broadcasts a vector data to every other process in the group.

Commonly used algorithms:

• Recursive Doubling (RD) Algorithm for small messages
• Ring Algorithm for large messages
Recursive Doubling (RD)
Ring
**RD** algorithm requires lesser number of communication steps  
→ the start-up costs are lower  
→ it is optimal for small and medium messages  

**Ring** scheme performs better for larger messages  
→ Because of near-neighbor communication pattern.
Collective Design Framework

Collective Algorithms

- Conventional Schemes
  - Pt2pt

- Single Leader Schemes
  - Pt2pt
  - Shmem
Conventional designs

- Not taking into account the hierarchy
  - introduced by the advent of multicore architectures.

- Not taking advantage of shared memory
  - for communication across cores residing on the same node. This could lead to very high network traffic
    - since every data-transfer could go across the network.

- Need to have single-leader-based hierarchical approaches
Single Leader approaches

Aggregation – Distribution

- Step 1: Data aggregation at the leader on each node
- Step 2: Inter leader exchanges ➔ Could be RD or Ring based on msg size
  - identical to the traditional clusters with single process per node
- Step 3: Data distribution within each node
Single Leader approaches: intra-node operations

Steps 1 and 3 are intra-node operations

→ **Point-to-point**

- data transfers are implemented on top of pt-to-pt MPI_Send and MPI_Recv operations
- the entire communication operation is serialized ➔ less performance

→ **Shared memory**

- leader sets up a shared memory buffer
- other processes read/write their contribution into this shared memory space **concurrently** ➔ could lead to better performance.
Single Leader algorithms on the AMD Barcelona Architecture
Single leader design

Pros compare to conventional schemes:

- Reduces network traffic
- Increase performance

However:

- Ignores the multi-core aspect of the machines
- Will not scale as the number of cores per node increases
  - Problem of memory contention ➔ lower scalability and poor performance
- Making multiple copies of messages
  - Limiting the performance as the message size increases
Problem Statement

- Be Multi-core and NUMA aware
  ◦ To achieve better performance and scalability
    ◦ With core counts and system sizes increase

- Exploit the differential memory access costs
  ◦ In NUMA based system
Proposed Collective Design Framework

- Collective Algorithms
  - Conventional Schemes
    - Pt2pt
  - Single Leader Schemes
    - Pt2pt
    - Shmem
  - Multi Leader Schemes
    - Pt2pt
    - Shmem
Multi-Leader based Algorithms (Step 1)
Multi-Leader based Algorithms (Step 2)
Multi-Leader based Algorithms (Step 3)
Performance of Multi-Leader schemes on large scale Multi-cores

4-Leader Point-to-point scheme outperforms the recursive doubling method on 1024 processes on the TACC Ranger
Performance of Multi-Leader schemes on large scale Multi-cores

Conventional Ring Algorithm performs better for larger messages
## Proposed Unified Scheme

<table>
<thead>
<tr>
<th></th>
<th>Intra-Node Mechanism</th>
<th>Inter-Leader Algorithm</th>
<th>Design</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small Messages</td>
<td>Point-to-Point</td>
<td>Recursive Doubling</td>
<td>Hierarchical</td>
</tr>
<tr>
<td>Medium Messages</td>
<td>Shared Memory</td>
<td>Recursive / Ring</td>
<td>Hierarchical</td>
</tr>
<tr>
<td>Large Messages</td>
<td>Point-to-Point</td>
<td>Ring</td>
<td>Conventional</td>
</tr>
</tbody>
</table>
Designing Non-blocking Broadcast with Collective Offload on InfiniBand Clusters: A Case Study with HPL

K. KANDALLA, H. SUBRAMONI, J. VIENNE, S. PAI RAIKAR, K. TOMKO, S. SUR, AND D. K. PANDA
Before MPI-3

- There exist only blocking collectives
- Application has to wait until collective operation completes
  - Limiting the performance
  - Portability that application might achieve

- Need to have non-blocking collectives in MPI-3
Host based approach in non-blocking collectives

- Used in libNBC
- Require user application developer to call MPI_Test
- Make a separate thread to progress the collective in the background
  - It usually have more latency than the blocking approaches
    - Because of creation and scheduling
  - Has low portability
    - We have to redo the tuning for the new platform
  - But has good communication and computation overlap
    - If it is tuned carefully
Desired Goals

1. Good latency \(\rightarrow\) At least as good as blocking versions
2. Good communication and computation overlap
3. Performance portability
Offload based non-blocking collectives: General Idea

- Offload the collective operations in to the InfiniBand adaptor/ NIC
- No need the intervention of host to progress the communication
- Host is needed to:
  - Create a task list
  - Each task could be receive/post/wait tasks
  - Send this list to NIC
- After send, host is free to continue its computations
  - Communication progress is guaranteed to be done by NIC
- Management Completion Queue
  - NIC sends a completion event to MCQ after each list is done
  - Host needs poll this queue during MPI_wait
Offload based non-blocking collectives: CORE-Direct

- Using CORE-Direct feature in the new NICs
  - ConnectX-2 Network Interfaces of Mellanox

- Arbitrary lists of send, receive and wait operations can be created

- These lists can then be posted to a work-request queue

- To be further processed by the network card
Offload based Broadcast collective

A tree based algorithm for bcast

⇒ Each node could be a root/leaf/intermediate
  ◦ Intermediate needs to be managed carefully

Challenge:

⇒ Task lists are created before even the broadcast operation begins
  ◦ ⇒ Need to know where exactly data will be when it arrives
  ◦ And then use that particular memory location to do the remaining send operation
Small and Medium Message Length Broadcasts

- Network-offload based k-nomial tree algorithm
- Most MPIs prepost buffers to NIC
  - To minimize the latency
- In this design, they mirror this list in lib
  - To know where exactly data will be
    - When it comes
  - When data comes, NIC asynchronously dequeue it
    - From its list
- This design still holds this information and will use it during remaining send tasks
Large Message Length Broadcasts

- Network-offload based scatter-allgather algorithm
  - Binomial exchange for the scatter phase and the ring algorithm for the allgather phase of the operation

- Dynamically register the user buffers and use RNR technique
  - For flow control and eliminating the need for buffer within the library

- RNR technique
  - Simplifies the design, Because we can directly use the registered user buffers to create the send/recv tasks
Algorithm Design Choices

- schedule the entire broadcast operation across the leader and the inter-node communicators

1) Flat-Offload (FO):
   - Use network-offload algorithms directly on the given communicator, without considering the node-level topology.
   - does not consider the node-level topology ↪ its latency is higher
   - current ConnectX-2 interface allows us to post task-lists of fixed sizes ↪ hard to scale
Algorithm Design Choices

2) Two-level-Offload-Host (TOH):

- Use network-offload based mechanisms for the leader exchange phase and complete the intra-node phases through send/recv during the MPI Wait operation.
- intra-node phases of the broadcast operation are done through shared-memory ➔ better latency for small messages
- library performs the intra node phase during the MPI Wait ➔ higher overheads due to the MPI Wait operation for large messages ➔ less overlap
Algorithm Design Choices

3) Two-level-Offload-Offload (TOO):
- Use network-offload algorithms for both the leader exchange and the intra-node phases.
  - Does not involve any intervention of the host ➔ deliver better overlap
  - intra-node phase relies on the network-loopback ➔ might not deliver the best communication latency

- ➔ TOH approach for small messages ➔ good communication latency
- ➔ TOH or TOO approaches for large messages ➔ depending on whether we want to optimize for latency or overlap
Performance Evaluation
A Novel Functional Partitioning Approach to Design High-Performance MPI-3 Non-Blocking Alltoallv Collective on Multi-core Systems

K. KANDALLA, H. SUBRAMONI, K. TOMKO, D. PEKUROVSKY AND D. K. PANDA
Hardware features like CORE-direct ➔

- Eliminates the need for the host to progress the communication.
- It also provides a low-level mechanism to design nonblocking collective algorithms
- ➔ several performance and scalability limitations
  - Separate Queue-Pairs (QPs) and Completion-Queues (CQs) for each communicator
    - large number of resources the HCA has to manage
  - This design’s communication performance is worse than the basic InfiniBand verbs-layer
  - Also drive the costs of supercomputing systems higher
New trend: Spare core to progress communication

With increasing number of compute cores per node

➔ the concept of improving concurrency through software-based approaches

**Problems:**

Involve additional memory resources and Expensive copy operations

◦ explicit data movement between the application tasks and the communication helper process
◦ overheads associated with additional copies

➔ limit the overall performance and scalability benefits
Desired Goals

1) Maximize computation / communication overlap

2) Eliminate the need for additional memory for intermediate buffering

3) Eliminate copy overheads between application processes and the spare core

4) Achieve memory and network scalability
Proposed design

Benefits:

➔ Eliminates the need for additional memory resources and expensive copy operations
  ◦ between the application processes and the CS

➔ Deliver near-perfect computation/communication overlap
  ◦ allows our servlet thread to seamlessly execute and progress the collective operations on behalf of the application tasks

➔ Completely user-level and is easily adoptable across supercomputing systems
Shared-Memory Based design

Requires one core per node to be available for the CS thread

CS mmaps the memory regions allocated by all the application processes that are executing on the same node

CS also registers these buffers with the InfiniBand network interface

- Servlet can directly execute data transfer operations on behalf of the application processes
  - without requiring additional memory for buffering data

- This design also eliminates the need for performing expensive copy operations to move the data between the application processes and the CS, before and after a collective operation
Shared-Memory Based design for non-blocking collective communication servlet

- process P0 on Node0 is sending data to process P10 on Node1 → CS independently transfer the data
- Exchange control information between the application processes and the CS
- Application encode the collective schedule in the form of a task-list, with each “task” including information about the specific send/recv operation → after it, app is free
- Pool of tasks in a separate shared-memory region
Communication protocol between servlets

Seq Num ➨ to ensure that data transfer operation happens ➨ only when
- the parent processes have entered the MPI library
- posted the collective operation

1. CS-0: RDMA-Write to update seq num
   - In Node1

2. CS-1: RDMA-Read to read data
   - From Node0

3. CS-1: RDMA-Write to update seq num
   - In Node0
Host-based/GOAL/Proposed Designs

(a) LibNBC-thread

(b) GOAL / kernel-GOAL

(c) Proposed Solution
256 processes/15 MPI processes in each node

Inserting MPI Test() calls within the compute loop, between the MPI Ialltoally and the MPI Wait operations to help progress the communication, for the default implementation in MVAPICH2
P3DFFT Execution time with 1,024 Processes
Designing Non-blocking Personalized Collectives with Near Perfect Overlap for RDMA-Enabled Clusters

HARI SUBRAMONI, AMMAR AHMAD AWAN, KHALED HAMIDOUCHE, DMITRY PEKUROVSKY, AKSHAY VENKATESH, SOURAV CHAKRABORTY, KAREN TOMKO, AND DHABALESWAR K. PANDA
Desired Goals

(1) Maximize computation / communication overlap

(2) Eliminate the need for MPI Test MPI Probe / MPI Iprobe calls to progress communication

(3) Achieve good communication latency and network scalability
Overall design-space for MPI-3 non-blocking collectives

<table>
<thead>
<tr>
<th>Metric</th>
<th>LibNBC</th>
<th>CORE-Direct</th>
<th>FP</th>
<th>Proposed Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Communication Latency</td>
<td>Good</td>
<td>Fair</td>
<td>Good</td>
<td>Good</td>
</tr>
<tr>
<td>Computation / Communication Overlap</td>
<td>Poor</td>
<td>Good</td>
<td>Good</td>
<td>Good</td>
</tr>
<tr>
<td>Network Scalability</td>
<td>Good</td>
<td>Poor</td>
<td>Good</td>
<td>Good</td>
</tr>
<tr>
<td>Availability of Cores for Compute</td>
<td>Poor</td>
<td>Good</td>
<td>Fair</td>
<td>Good</td>
</tr>
</tbody>
</table>
Design alternatives for non-blocking collectives

(a) LibNBC
(b) Functional Partitioning
(c) Proposed Solution
Design of RDMA-Aware Non-blocking Personalized Collectives

Current collective designs in high performance MPI libraries are already using RDMA

**Problem:**

data transfer is still based on underlying two-sided point-to-point communication
  ◦ require CPU intervention to progress control messages and completion

→ we need operations that rely on direct RDMA / one-sided semantics
Main challenges in designing high performance and scalable RDMA-Aware NBC

(1) efficiently exchanging the tuple of \( <\text{remote memory address}, \text{remote rkey}> \) required to perform RDMA operations (RDMA-Write and RDMA-Read) between processes

(2) exchanging data using appropriate RDMA primitives over the network to remote peers

(3) notifying the remote peer of the completion of the data transfer.
Design of Non-blocking RDMA-Aware All-to-all

There are several algorithms available to implement the All-to-all

most researchers agree that for large message sizes and large system sizes, one needs some sort of a pairwise exchange algorithm

◦ where all processes talk directly to each other without any intermediate process routing the messages on their behalf.

◦ ➞ extremely amenable to designing using basic RDMA operations
Design of Non-blocking RDMA-Aware All-to-all
Some optimizations

Caching Mechanism to Avoid MPI Allgather

caching mechanism to store the tuple from all processes
  ◦ avoid invoking the Allgather operation in each iteration of the All-to-all

After registering the buffers with the IB HCA
  ➔ processes participating in the All-to-all will compare the tuple and cached tuple

Due to the “IB memory registration cache” in MVAPICH2
  ➔ highly likely that if the addresses of the send / receive buffer passed by the application remains unchanged, the remote key will also be the same.

After the local comparison phase ➔ MPI Allreduce operation to see if all processes had a “cache hit” ➔ If so, then they skip the MPI Allgather
Design of Non-blocking RDMA-Aware All-to-one and One-to-all

result of extensive performance tuning done for collective operations ➔ the “direct”

algorithm, where all processes send / receive data directly to the “root” of the All-to-one / One-to-all operation, leads to the best performance

➔ very amenable to designing using basic RDMA operations as there are no intermediate steps in the communication that require intervention either from a process/thread or a hardware collective offload engine like Core-Direct.
Design of Non-blocking RDMA-Aware All-to-one and One-to-all

1. processes register either the send or the receive buffer passed by the application with the IB HCA to obtain the “remote rkey”.
   - All-to-one, the root will register the receive buffer / non-root processes will register the send buffer.
   - One-to-all on the other hand, the root will register the send buffer while the non-root processes will register the receive buffer.

2. Broadcast operation of 12 bytes to inform the non-root processes of its tuple
   - use the hardware multicast based MPI Bcast in MVAPICH2
     - achieve good scalability and performance.
Design of Non-blocking RDMA-Aware All-to-one and One-to-all

3. initiate the RDMA operations for transferring the data non-root processes:
   ◦ All-to-one ➔ RDMA-Write operation
   ◦ One-to-all ➔ RDMA-Read operation

   ➔ reduce the load on the IB HCA at the root of the collective

4. posting completion notification to the root
   ➔ RDMA-Write primitive for the completion notification for both collective patterns a
     as it is just a one byte transfer.
Design of Non-blocking RDMA-Aware All-to-all

1. each process first registers the send / receive buffers passed by the application with the IB HCA to obtain the “remote rkey”
   ◦ hide the cost of registration by caching the registered entries
   ◦ Each process also allocates and registers (with the IB HCA) a temporary buffer to check for completion of data transfers from remote processes

2. small message (24 byte) MPI Allgather is then performed between all processes taking part in the All-to-all operation to collect the tuple of < remote memory address, remote rkey > from all processes and notify them of its completion.
Design of Non-blocking RDMA-Aware All-to-all

3. each process schedules data transfer and completion notification operations for each peer on a remote node by posting RDMA-Read or RDMA-Write operations to the IB HCA

Intra node connections:
- Shared memory communication channel for small messages ➔ less latency
- Loopback communication channel through the IB HCA for large messages ➔ more overlap
Performance of P3DFFT for different NBC designs

(a) Small scale runs

(b) Large scale runs
Thank you!