IEEE Transactions on Parallel and Distributed Systems, Vol. xx, No. xx, Xxx 2018

Exploiting Hardware Multicast and GPUDirect RDMA for Efficient Broadcast

Ching-Hsiang Chu, *Student Member, IEEE,* Xiaoyi Lu, Ammar A. Awan, Hari Subramoni, Bracy Elton, *Senior Member, IEEE* and Dhabaleswar K. Panda, *Fellow, IEEE*

Abstract—Broadcast is a widely used operation in many streaming and deep learning applications to disseminate large amounts of data on emerging heterogeneous High-Performance Computing (HPC) systems. However, traditional broadcast schemes do not fully utilize hardware features for Graphics Processing Unit (GPU)-based applications. In this paper, a model-oriented analysis is presented to identify performance bottlenecks of existing broadcast schemes on GPU clusters. Next, streaming-based broadcast schemes are proposed to exploit InfiniBand hardware multicast (IB-MCAST) and NVIDIA GPUDirect technology for efficient message transmission. The proposed designs are evaluated in the context of using Message Passing Interface (MPI) based benchmarks and applications. The experimental results indicate improved scalability and up to 82% reduction of latency compared to the state-of-the-art solutions in the benchmark-level evaluation. Furthermore, compared to the state-of-the-art, the proposed design yields stable higher throughput for a synthetic streaming workload, and 1.3x faster training time for a deep learning framework.

Index Terms—Broadcast, Deep Learning, Hardware Multicast, GPU, GPUDirect RDMA, Heterogeneous Broadcast, Streaming

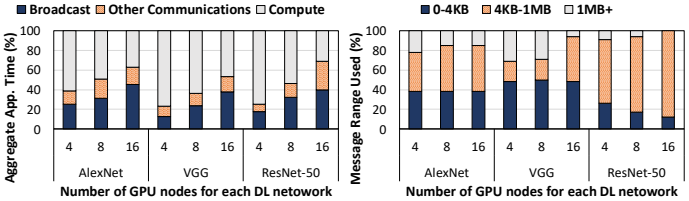
1 Introduction

Emerging high-performance computing (HPC) systems are marked by two factors: 1) the usage of accelerators like general purpose graphics processing units (GPGPUs) to boost their computing capabilities, and 2) high-performance commodity interconnects such as InfiniBand (IB) to push frontier of performance and scalability. As a result, numerous HPC applications, runtimes, and frameworks are adopting the massive parallelism computing power of GPUs [1], [2], [3], [4], [5], [6].

Many GPU-based applications behave like a streaming application [1], [7]. Specifically, a GPU-based streaming application is generally comprised of two concurrent phases: 1) A compute-intensive phase that is executed in parallel by multiple workers on GPU nodes, with the main computation executed on GPUs typically along with traffic between the host CPU and the GPUs using PCIe or resources, and 2) A communication phase where applications rely heavily on broadcast-type operations to move data from a single source to multiple GPU-based computing workers.

Deep Learning (DL) includes GPU-based HPC applications, which can benefit from accelerated computation and communication. DL applications have been widely used to model and solve various real-world problems in areas such as bioinformatics, computer vision, and natural language processing. As DL applications require processing increasingly large datasets, it is becoming common for them to utilize GPU-enabled HPC clusters. For example, many DL frameworks, such as Caffe [8] and its variants [2], [9], Facebook Caffe 2 [10], Google TensorFlow [11], Microsoft Cognitive ToolKit (CNTK) [12], have been designed to easily and efficiently conduct the time-consuming DL training process on GPU-enabled HPC clusters.

In streaming and DL applications, collective operations such as *broadcast* are often involved for exchanging large amounts of data among GPUs across nodes on HPC clusters. CA-CNTK framework [13], for example, relies heavily on inter- and intra-node broadcast operations to exchange data among all GPUs. As shown in Figure 1(a) (see Section 7 for configuration information), we can see a significant increase of communication time on broadcast operations as the scale increased. Moreover, as can be seen in Figure 1(b), more than half of message sizes of broadcast operations are higher than 4 KB and the maximum message size of broadcast can be as large as around 128 MB here. This implies a urgent need to optimize broadcast operations for large messages in order to conduct efficient large-scale DL training process. Moreover, applications can assign GPU nodes to different groups, e.g., communication groups in the Message Passing Interface (MPI), with each group working on different sets of data. Such application scenarios require concurrently performing multi-source broadcast operations over an entire cluster—adding challenges for achieving high performance for large-scale jobs. More efficient and scalable multi-source broadcast operations can help improve the performance of streaming and DL applications on GPU clusters.



(a) Application Time (b) Message Sizes Used for Broadcast

Fig. 1. Profiling of training process for image classification using CA-CNTK framework [13] with ImageNet Dataset [14] on a GPU cluster.

- *C.-H. Chu, X. Lu, A. A. Awan, H. Subramoni, D. K. Panda are with the Department of Computer Science and Engineering, The Ohio State University, Columbus, OH 43210–1277.*
  E-mail: {chu.368, lu.932, awan.10, subramoni.1, panda.2}@osu.edu
- *Bracy Elton is with Engility Corporation, WPAFB, OH 45433–7802.*
  E-mail: bracy.elton@engility.com

However, traditional broadcast schemes, e.g., Ring and K-nomial-based, are not well-optimized for large-scale GPU-based DL applications. In these schemes, the number of communication steps grows as the number of participating GPU processes grows, eventually causing scalability issues at large scales. Although highly scalable InfiniBand hardware-based multicast (IB-MCAST) [15] technology has been adopted in HPC middleware to enhance the scalability on traditional homogeneous HPC clusters [16], [17], [18], [19], they have not yet been optimized for large message multi-source broadcast scenarios, situations that appear commonly in DL applications. Also, heterogeneous broadcast support is required for streaming applications on HPC cluster systems. These issues lead to the following broad challenges:

- Accurately modeling, and analyzing fundamental performance bottlenecks of existing broadcast operations on GPUs
- Determining techniques to leverage IB-MCAST and other GPU advanced features, such as GPUDirect RDMA (GDR) to design efficient and scalable broadcast with large messages on GPU clusters
- Achieving high overlap and scalability for multi-source broadcast operations, so that multi-source broadcast operation latency increases sublinearly (near constant) commensurate with an increasing number of sources
- Determining attainable theoretical and practical performance benefits for deep learning applications

To begin exploring these challenges, this paper first models existing broadcast schemes, i.e., Ring, K-nomial, and IB-MCAST, and analyzes their performance bottlenecks on GPU clusters. Based on this analysis, we propose a novel broadcast design built on message streaming to better exploit IB-MCAST and GDR technologies for efficient large message transfers. The proposed design can provide high overlap for multi-source broadcast operations. The experimental results show that our proposed design can attain 82% latency reduction compared to state-of-the-art solutions in a benchmark-level evaluation. The results also show a 24% performance improvement in CNTK deep learning frameworks across varied training data sets. Our model validation shows that the proposed analytical model matches experimental results within 10%. Our model also predicts that the performance of our proposed design achieves near-constant latency for a single broadcast operation with an increasing number of participating GPU processes, while the latency of our proposed broadcast design increases sublinearly with an increasing number of broadcast sources. To summarize, towards improving broadcast operations, this paper makes the following key contributions:

- Provides and evaluates analytical models to capture essential performance behavior of alternative broadcast schemes on GPU clusters (Section 3)
- Proposes efficient and scalable zero-copy heterogeneous broadcast operations by taking advantage of IB-MCAST and IB Scatter-Gather-List (SGL) features, and NVIDIA GDR technology (Section 4)
- Proposes advanced design of GPU-based multi-source streaming broadcast operation for large-size GPU-to-GPU message transfers (Section 5)
- Proposes efficient intra-node topology-aware broadcast operations by leveraging CUDA Inter-Process Communication (IPC) features for multi-GPU systems (Section 6)
- Evaluates performance and analyzes the proposed design for deep learning models on a real-world GPU-enabled InfiniBand cluster (Section 7)

2 Background

In this section, we review state-of-the-art hardware platforms and important software features available in InfiniBand (IB) host channel adapters (HCA) and switches, and NVIDIA GPUs.

2.1 InfiniBand Features

InfiniBand is one of the most widely deployed commodity network technologies in HPC systems [15]. Similar to TCP/IP protocol, IB offers four transport modes: Reliable Connection (RC), Reliable Datagram (RD), Unreliable Connection (UC), and Unreliable Datagram (UD). Most importantly, it allows for using remote direct memory access (RDMA) technology to significantly improve the performance of network communication. Notable features related to this paper are described as follows.

InfiniBand provides support for a so-called **scatter-gather list (SGL)** in allowing data transfer operations from multiple memory locations in a single operation. While using channel semantics, InfiniBand provides support for *gathering* data from multiple memory locations within a node and transferring it to a receiver in one operation. Similarly, the receiver side also has the flexibility to *scatter* incoming data to different, possibly non-contiguous, memory locations. Beside common communication patterns like one-to-one and one-to-all, multicast is a one-to-many communication pattern. Absent hardware support for it, multicast operations are typically achieved via multiple point-to-point communications, i.e., *software multicast*. The **IB hardware multicast (IB-MCAST)** feature allows for issuing a single multicast send operation with the IB switch itself handling the one-to-many operation *in hardware*. Compared to a software-implemented multicast operation, employing hardware multicast technology can significantly reduce network traffic and provide high scalability.

2.2 NVIDIA GPUs and GPUDirect Technology

NVIDIA graphics processing units (GPUs) have gained significant attraction in the HPC community in recent years. An ever increasing number of redesigned classical HPC applications have emerged that take advantage of massively parallel GPU hardware. In general, GPUs are connected as peripheral devices on the Peripheral Component Interconnect Express (PCIe) bus or NVLink [20]. The NVIDIA Compute Unified Device Architecture (CUDA) is the default parallel programming platform and programming model for NVIDIA GPUs. Successive generations of CUDA have introduced new features to leverage the potential of next-generation hardware features in GPUs. CUDA 4.0 introduced Unified Virtual Addressing (UVA), which exposes a unified address space for CPU and GPU memory. In

addition, the GPUDirect family of features [21], alleviating the overhead of an extra data transfer operation otherwise required to push data from GPU memory into the network.

To address limitations of intra-node Peer-to-Peer (P2P) communication between GPUs, Inter-Process Communication (IPC) technology was introduced in CUDA 4.1. CUDA IPC allows direct GPU-to-GPU memory access without involving the CPU for assistance. Furthermore, the GPUDirect RDMA (GDR) mechanism introduced in CUDA 5.0 enables third-party PCIe devices to directly access GPU-resident data, completely bypassing CPU memory. This provides a path for moving data to/from GPU device memory over an IB network in a way that completely bypasses CPU memory. This also helps in reducing PCIe resource consumption by decreasing the number of PCIe links involved.

3 Modeling and Analysis of Existing Broadcast Algorithms on GPU Clusters

Existing broadcast algorithms for GPU-resident data are typically host-based algorithms. While host-based broadcast algorithms have been well-studied, accelerators, such as GPUs, pose features and issues calling for additional investigation and consideration. In this section, we provide detailed models and analysis thereof for the most commonly used broadcast algorithms. Furthermore, we identify limitations of existing broadcast schemes and present how broadcast operations on GPU clusters may benefit from algorithms employing IB-MCAST and GDR features.

3.1 Existing Broadcast Schemes and Their Performance Models

In applying existing broadcast algorithms to GPU-resident data, the main issue regards how an HCA retrieves such data. Existing communication libraries typically move data from GPU to CPU host memory, and then apply host-based broadcast algorithms, which are well-studied in the literature [22], [23], [24], [25], [26]. However, simply employing existing host-based broadcast algorithms may negatively impact performance. There are two typical ways for supporting GPU-aware broadcast with existing algorithms: (1) leveraging NVIDIA GDR-like technology to allow an HCA to read/write data directly from/to GPU memory; (2) staging GPU-resident data to CPU host memory. In this section, we discuss the most popular GPU-aware broadcast algorithms, i.e., ring-based and K-nomial-based ones, from the perspective of the broadcast sender. Table 1 summarizes notations used herein.

3.1.1 Ring-based Broadcast Algorithm Performance Model

In a ring-based broadcast algorithm, processes form a logical ring structure. Beginning with the broadcast sending process, each process sends data to its logical neighbor. Overall, this takes $n-1$ steps. In each step, the process takes $(t_s+\frac{M}{B_G})$ time to push data to the network. Then, naively applying GDR technology in ring-based algorithms, the time for such an operation is given by

$$T_{(Bcast\_Ring\_GDR)} = (n-1) \times (t_s + \frac{M}{B_G}). \quad (1)$$

To further improve performance, i.e., reduce the time for the operation, one can divide a message into smaller chunks and send those in a pipelined fashion. Data communication can then be overlapped across the ring, the cost then being,

$$T_{(Bcast\_Ring\_GDR\_Pipeline)} = (\frac{M}{C} + (n-2)) \times (t_s + \frac{C}{B_G}), \quad (2)$$

where each step then takes time $t_s + \frac{C}{B_G}$ to push data to the network, with $\frac{M}{C} + (n-2)$ steps because the sender requires $\frac{M}{C}$ steps to push all data chunks to its neighbor and it takes $(n-2)$ steps for the last data chunk to travel through all processes. Compared to the pure GDR approach (Equation 1), the number of steps increases from $(n-1)$ to $\frac{M}{C} + (n-2)$ while the cost (time) per step decreases from $\frac{M}{B_G}$ to $\frac{C}{B_G}$. Indeed, it is important to tune the data chunk size (C) to optimize performance, minimizing latency and maximizing throughput.

As the bandwidth between a CPU and an HCA (B_H) is typically higher than the bandwidth between a GPU and an HCA (B_G) in current architecture, another common approach is to stage GPU data through CPU host memory. However, this process incurs overhead for moving data between GPU and host memory. Therefore, a performance model for the scheme without pipelining is as follows:

$$T_{(Bcast\_Ring\_Staging)} = \frac{M}{B_{H2D}} + (n-1) \times (t_s + \frac{M}{B_H}). \quad (3)$$

TABLE 1
Notations for the Analytical Models

Name	Description	Unit
n	Number of processes	N/A
m	Number of broadcast sources	N/A
t_s	Set up time for sending data	sec
$t_o(n)$	Overhead for issuing an IB-MCAST packet	sec
M	Message size	bytes
C	Size of a data chunk	bytes
U	Maximum Transmission Unit for IB-MCAST	bytes
B_H	Bandwidth of reading Host memory	bytes/sec
B_G	Bandwidth of reading GPU memory (GDR read)	bytes/sec
B_{H2D}	Bandwidth between Host and GPU memory	bytes/sec

3.1.2 K-nomial-based Broadcast Algorithm Performance Model

The next algorithm examined is the K-nomial algorithm, the most popular broadcast algorithm. When K is two, this is the well-known Binomial algorithm. The main difference between ring-based and K-nomial algorithms is that the latter reduce the communication cost from being linear to logarithmic [22]. Therefore, similar to ring-based algorithms, performance models for K-nomial-based broadcast algorithms for GPU-resident data can be represented as follows:

$$T_{(Bcast\_K-nomial\_GDR)} = \lceil \log_k n \rceil \times (t_s + \frac{M}{B_G}), \quad (4)$$

$$T_{(Bcast\_K-nomial\_Staging)} = \frac{M}{B_{H2D}} + \lceil \log_k n \rceil \times (t_s + \frac{M}{B_H}), \quad (5)$$

$$T_{(Bcast\_K-nomial\_GDR\_Pipeline)} = (\frac{M}{C} \times \lceil \log_k n \rceil) \times (t_s + \frac{C}{B_G}). \quad (6)$$

Note that the K-nomial algorithm with GDR and pipelining still requires $\lceil \log_k n \rceil$ steps to allow the sender to push one data chunk into the network, with such $\frac{M}{C}$ data chunks.

TABLE 2
Comparison of broadcast schemes in the literature

Bibliography Reference	Scheme	Pipeline	Scalability	Heterogeneous Support	GPU-Awareness Capability	
					GDR	Staging through Host
[27]	Ring	Y	Low	N	Y	N
[28]	Ring	Y	Low	N	Y	Y
[24]	Ring+Binomial	Y	Medium	N	N	N
[25]	Binomial	Y	Medium	N	N	N
[29]	Ring	Y	Medium	Y	Y	Y
[17]	IB-MCAST+Ring	N	Medium	N	N	N
[18]	IB-MCAST	N	High	N	N	N
[19]	IB-MCAST	N	High	N	Y	N
[30]	IB-MCAST	N	High	Y	Y	N
[31]	IB-MCAST	Y	High	N	Y	Y
Proposed design	IB-MCAST	Y	High	Y	Y	Y

3.1.3 IB-MCAST-based Broadcast Algorithm Performance Model

In IB-MCAST-based broadcast schemes, the communication cost is much less affected by the number of nodes than it is by the IB Maximum Transmission Unit (MTU) for UD (U) [19], as discussed below in Section 3.2. Therefore, it takes $\frac{M}{U}$ steps to push data to the network. Based on our observations, there is a small overhead ($t_o(n)$), discussed in Section 8, for issuing an IB-MCAST operation. A performance model for broadcast operations based on IB-MCAST and GDR is then given by

$$T_{Bcast\_MCAST\_GDR} = \frac{M}{U} \times (t_s + t_o(n) + \frac{U}{B_G}). \quad (7)$$

3.2 Analysis

The importance of scalability of broadcast-like collectives for GPU-resident data increases commensurate with GPU cluster system size. Table 2 summarizes key differences among commonly used broadcast schemes in the literature. From the performance models without IB-MCAST shown in Section 3.1, we can observe that even when employing staging or pipelining mechanisms, the *number of nodes involved* fundamentally impacts the performance of ring-based and K-nomial-based broadcast algorithms. However, in the IB-MCAST-based broadcast algorithm, the number of nodes involved has little-to-no impact on performance. Therefore, *employing IB-MCAST provides an opportunity for a scalable broadcast operation* not only for traditional CPU-based but also for GPU-enabled clusters.

The main performance bottleneck of the existing IB-MCAST-based broadcast scheme for GPU data is the PCIe P2P read limit of GDR [19], [32], B_G, as defined in Table 1, and the IB MTU limit. As a result, existing IB-MCAST-based broadcast algorithms do not presently yield good performance for large messages, as notably commonly appear in streaming and deep learning applications.

4 Designing Zero-Copy Heterogeneous Broadcast for Streaming Applications

As indicated in Section 1, streaming applications are typically comprised of compute nodes and a source component that generates data in real time (live source or staged through disk). To take advantage of the computing power of GPUs, the data needs to be continuously broadcast to multiple GPU computing nodes. However, state-of-the-art approaches for the broadcast primitive typically support only homogeneous operations, i.e., both the source and destination need to be either in the host CPU or in GPU memory. As a result, moving data between host and GPU memory is needed to support heterogeneous operations. This presents potential bottlenecks. To alleviate these bottlenecks, we propose efficient heterogeneous broadcast designs by taking the advantage of IB and NVIDIA GDR hardware features, as described below.

4.1 Scatter List (SL)-based Heterogeneous Broadcast

Any data movement operation over networks is composed of *control information* and *the data payload*. Generally, the control information is always generated and stored in host memory while the data payload can be in GPU memory. To directly write/read the control information to host and the data payload to GPU memory in one single operation, respectively, communication libraries usually rely on the IB SGL feature and NVIDIA GDR capabilities [19], [33], as described in Section 2.

When performing heterogeneous broadcast operations, the source data and control information typically both reside in host memory. On the receiver side, data and control information need to reside in GPU and host memory, respectively. One can also perform a broadcast operation from GPU memory to remote CPU host memory; however, this is not commonly used in practice.

As mentioned earlier, existing approaches that exploit the IB-MCAST feature only support a homogeneous broadcast scheme. Thus, additional data transfer operations are required either before GPU-based or after host-based broadcasts [19], [34]. To avoid these costly data transfer operations between host and GPU memory, we present an IB scatter-list (SL)-based design, which provides an efficient broadcast algorithm, combining both GDR and IB-MCAST features. Specifically, on the receiver side, when posting receive requests to the IB HCA, the communication library specifies host and GPU memory addresses for control information and data, respectively. As shown in Figure 2, when data arrives, the IB HCA performs a scatter operation to write data directly to the registered GPU memory space—without staging through host memory since the GDR feature enables third-party devices to access GPU memory directly via the PCIe bus. (Note that the receiver GPU memory needs to be registered in order to allow the IB HCA direct access to it.) To avoid the overhead of memory registration, removing it from the critical communication path of a broadcast operation, we intercept the MPI memory allocation application program interfaces (APIs) and transparently perform memory registrations. Further, we maintain a registration cache to reduce registration overhead.

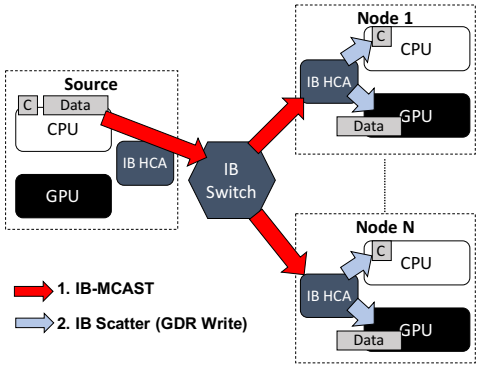


Fig. 2. Proposed SL-based Heterogeneous broadcast

4.2 Zero-copy Multicast Scheme

Generally, on IB clusters, a communication library always registers and posts pre-allocated buffers to the HCA, i.e., *intermediate buffers*, to ensure that any unexpected network packets can be successfully handled. This enables the sender and receiver to progress asynchronously. However, this mechanism requires additional data transfer operations from these *intermediate buffers* to user-specified buffers on the receiver side. To perform a zero-copy multicast operation, the sender must wait until the receiver has registered and posted the user-specified buffers to ensure that IB-MCAST packets can be received successfully. Existing zero-copy schemes either employ barrier operations, e.g., MPI_Barrier, to synchronize between sender and its receivers, or apply a hybrid scheme for retransmitting lost IB-MCAST packets [17].

Here, we propose a zero-copy scheme that leverages one-sided communications [35], [36] to perform light-weight sender-receiver synchronization, followed by the proposed SL-based broadcast operation. The proposed scheme appears as Algorithm 1. In Algorithm 1, lines 8-15 represent the sender side. First, the sender checks whether all receivers have posted the buffers (lines 8-10) with **mcast_prepost_status** being a vector exposed to all receivers for performing one-sided operations. Next, the sender multicasts the data (lines 11-15). On the receiver side (lines 16-28), a receiver first posts the user-specified buffer to allow its HCA to directly write data to it without explicit copy operations (lines 16-22). Finally, the receiver notifies sender of the completion of post-processing by using one-sided operation (line 25) and waits for the IB-MCAST packets. Note that this zero-copy scheme can be applied to heterogeneous as well as homogeneous broadcast operations. Since the IB-MCAST feature is based on UD transport, i.e., a transport protocol that does not provide reliability support, reliability must be handled via software. In this paper, we focus on the broadcast design itself with reliability supported by applying the low-overhead scheme proposed in [37].

5 Designing Streaming Broadcast for Deep Learning

With deep learning applications requiring processing of large datasets, it is common practice for such applications to distribute workloads to multiple GPUs across multiple nodes in clusters. Broadcast operations are employed to handle the workload distribution and involve multiple concurrent broadcast operations among different communication groups, e.g., MPI communicators. And a particular

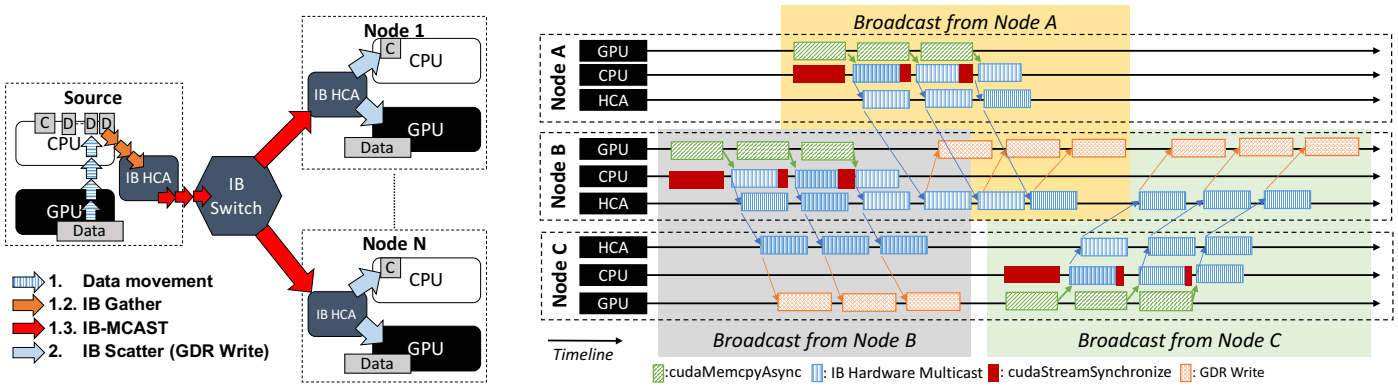
Algorithm 1 Proposed zero-copy broadcast scheme

```
1:  Definition of Variables (with respect to each process)
2:      buf : the user-specified buffer to be broadcast.
3:      size : length of buf in byte.
4:      seqnum : latest sequence number of IB-MCAST.
5:      UD_MTU : Unreliable Datagram Maximum Transmission Unit
6:      mcast_prepost_status : the latest sequence number of MCAST
7:      packet is pre-posed on the broadcast receiver sides
    ON THE BROADCAST SENDER SIDE:
8:      while ∀ s ∈ mcast_prepost_status < seqnum do
9:          /* Wait for broadcast receivers to update their status */
10:     end while
11:     bindex ← 0
12:     while bindex < size do
13:         mcast_send(buf+bindex)
14:         bindex += UD_MTU
15:     end while
    ON THE BROADCAST RECEIVER SIDE:
16:     /* Prepost the user buffer to receive MCAST packets */
17:     bindex ← 0
18:     while bindex < size do
19:         post_mcast_pkt(buf+bindex)
20:         bindex += UD_MTU
21:         seqnum++
22:     end while
23:     /* Notify the broadcast sender by using the light-weight
24:      * true one-sided operation */
25:     OneSided_Put(&seqnum, ..., bcast_sender, ...)
26:     while mcast_recv(buf,...) do
27:         /* Wait until all MCAST packets are received */
28:     end while
```

GPU may participate in multiple communications groups. In an efficient distribution scheme, multiple broadcast operations are issued concurrently in different communications groups. However, although there is a potential for overlapping broadcast operations involving different communication groups, existing broadcast schemes rather apply different algorithms based on point-to-point communications as discussed in Section 3, and in those schemes, each broadcast operation generally proceeds sequentially. Furthermore, we find that scalability of existing broadcast algorithms is limited as discussed in Section 3.

5.1 Streaming Broadcast Design

As mentioned in Section 3.2, the PCIe P2P read limitation brings significant overhead to GDR read operations, which, in turn, can become a significant bottleneck for performing inter-node GPU-to-GPU broadcast operations when using GDR features. To address this issue, we propose a two-step strategy as illustrated in Figure 3(a): (1) move the data to host memory, and (2) leverage the high-performance IB-MCAST-based broadcast algorithm as presented in Section 4, where the IB-MCAST sends data in host memory to GPU-resident receiver memory. Note that the first step would be expensive were we to naively perform the device-to-host (D2H) data transfer, i.e., using *cudaMemcpy*. To help hide the cost of D2H copy operations, we further propose dividing a message into multiple smaller chunks and issuing the associated D2H copy operations asynchronously, i.e., via *cudaMemcpyAsync*. Moreover, this intermediate host buffer is "pinned" by the GPU to achieve higher data transfer rates. Finally, the sender broadcasts the chunks when they are ready. In this way, these two steps work in a (software) pipelined fashion and can be overlapped. This avoids the P2P read limit and also leverages the high-performance IB-MCAST capability and employs GPUDirect



(a) Overview

(b) Design details with an example

Fig. 3. Proposed streaming broadcast on GPU clusters

features. When processing a large message as multiple data chunks, we collectively treat them as incoming streaming data in the broadcast kernel, with such processing herein termed, "streaming broadcast".

In deep learning applications, concurrent large message broadcast operations commonly occur across differing groups of nodes. However, since deep learning applications typically involve blocking-mode broadcast operations, e.g., MPI_Bcast, broadcast communications among different groups of nodes do not readily occur concurrently, though there is potential for using non-blocking broadcasts (this approach would require application-level changes). To achieve overlapping of broadcast operations across differing MPI communications groups (without modifying the applications), one could implement a blocking broadcast with multiple non-blocking point-to-point operations. However, scalability of such an approach is limited as discussed in Section 3. By using IB-MCAST, a single broadcast operation can return immediately once the IB-MCAST request is posted to the IB HCA. This allows a broadcast sender to return and participate in the next broadcast call once the current broadcast data is posted. This provides an opportunity for multiple broadcast operations to proceed concurrently and for the overall blocking broadcast to be implemented in the proposed pipelined fashion without application-level changes. Figure 3(b) exhibits an example with three GPU nodes (Nodes A, B, and C) involved in broadcast operations in three different groups. The arrow lines indicate the data flow. Rectangles represent data chunks, where three chunks are used for a single broadcast in the example. In each node, we assume one auxiliary process runs on the CPU host to facilitate communications. Note that while there could be more nodes involved in broadcast groups, for illustrative proposes, we show only three nodes.

We first consider the single broadcast from Node B to C. As soon as the first chunk is copied to Host memory, the CPU can begin continuously posting IB-MCAST requests to HCAs, which will gather and send out the data. Within a node, we see two levels of overlapping in a broadcast operation on the sender side: (1) moving data from GPU to host memory is overlapped with moving data from CPU host memory to an HCA, and (2) moving data between GPU and host memory and from host memory to an HCA also overlaps with communication over IB via IB-MCAST operations. On the broadcast receiver side, an HCA leverages the GDR feature to write incoming data directly to GPU buffers, notably bypassing CPU memory and also HCA-to-host and host-to-GPU PCIe paths. Therefore, the CPU is only responsible for polling the IB completion queue (CQ) to ensure that the IB-MCAST packets are received successfully. In this way, the receiver side activity also overlaps with the sender side, as depicted in Figure 3(b). Next, we consider a multi-source broadcast scenario, where Node A performs a broadcast to Node B while Node B is still performing broadcast to Node C in a different MPI communications group. In Figure 3(b), Node B can return immediately once all IB-MCAST requests are posted to HCAs. Therefore, once the HCA of Node B is free, that HCA can receive IB-MCAST data from Node A while another broadcast operation (Node B to C) occurs (concurrently). Similarly, the Node C to B broadcast can overlap with the Node A to B broadcast.

Algorithm 2 details the proposed streaming broadcast scheme on the sender side. The receiver side scheme remains the same as Algorithm 1. Multiple ***cudaMemcpyAsync*** calls are used (lines 10, 17, and 23) to asynchronously perform the data transfers between host and GPU memory. ***cudaStreamSynchronize*** calls are used (lines 15 and 21) to ensure the completion of data transfers when the data is required to be multicast. Specifically, the first asynchronous data transfer (line 10) can be overlapped with sender-receiver synchronization (lines 12–14). Moreover, subsequent asynchronous data transfers (lines 17 and 23) can be overlapped with IB-MCAST operations (line 27). Remaining parts are the zero-copy scheme presented in Algorithm 1.

5.2 Performance Models for the Proposed Design

The proposed design avoids the GDR read limit by staging GPU data through CPU host memory. Moreover, the pipelined design (with a fine-tuned data chunk size) hides the cost of GPU-to-host data transfer operations. This suggests a timing performance model for the proposed design with a single broadcast operation, as given by

$$T_{Bcast\_MCAST\_Staging\_Pipeline} = \frac{C}{B_{H2D}} + \frac{M}{U} \times (t_s + t_o(n) + \frac{U}{B_H}). \quad (8)$$

In the multi-source broadcast scenario, suppose there are m broadcast sources and further suppose there is a fraction of

Algorithm 2 Proposed streaming broadcast

```
1: Definition of Variables (with respect to each process)
2:   buf: the user-specified GPU buffer to be broadcast.
3:   mbuf: the pre-allocated host buffer used to perform IB-MCAST.
4:   cSize: the pre-defined chunk size in byte.
5:   copiedIndex: the record of the current index of copied data.
6:   strm: the pre-defined CUDA stream to perform data movement
7:   between Host and GPU memory.
   ON THE BROADCAST SENDER SIDE:
8:   mIndex ← 0
9:   copiedIndex ← 0
10:  cudaMemcpyAsync(mbuf, buf, cSize,
11:                  cudaMemcpyDeviceToHost,strm)
12:  while ∀ s ∈ mcast_prepost_status < seqnum do
13:    /* Wait for broadcast receivers to update their status */
14:  end while
15:  cudaStreamSynchronize(strm)
16:  copiedindex ← copiedindex + csize
17:  cudaMemcpyAsync(mbuf+copiedIndex, buf+copiedIndex,
18:                  cSize, cudaMemcpyDeviceToHost,strm)
19:  while mIndex < size do
20:    if mIndex < copiedIndex then
21:      cudaStreamSynchronize(strm)
22:      copiedindex ← copiedindex + csize
23:      cudaMemcpyAsync(mbuf+copiedIndex,
24:                      buf+copiedIndex, cSize,
25:                      cudaMemcpyDeviceToHost,strm)
26:    end if
27:    mcast_send(mbuf+mIndex)
28:    mIndex += UD_MTU
29:  end while
   ON THE BROADCAST RECEIVER SIDE:
30:  Same as in Algorithm 1
```

OP overlap among broadcast operations, then the cost of a multi-source broadcast scheme becomes $(1-OP)\times m$ times the cost of the corresponding single broadcast operation. Therefore, the timing performance model of the proposed design for the multi-source broadcast scenario can be represented as follows:

$$T_{M-Bcast\_MCAST\_Staging\_Pipeline} = \quad (9)$$
$$(1-OP)\times m\times(\frac{C}{B_{H2D}}+\frac{M}{U}\times(t_s+t_o(n)+\frac{U}{B_H}))\,.$$

And also in the multi-source broadcast scenario, we can similarly determine timing performance models for other broadcast schemes as illustrated in Section 3. Evaluation of the models is discussed in Section 8.

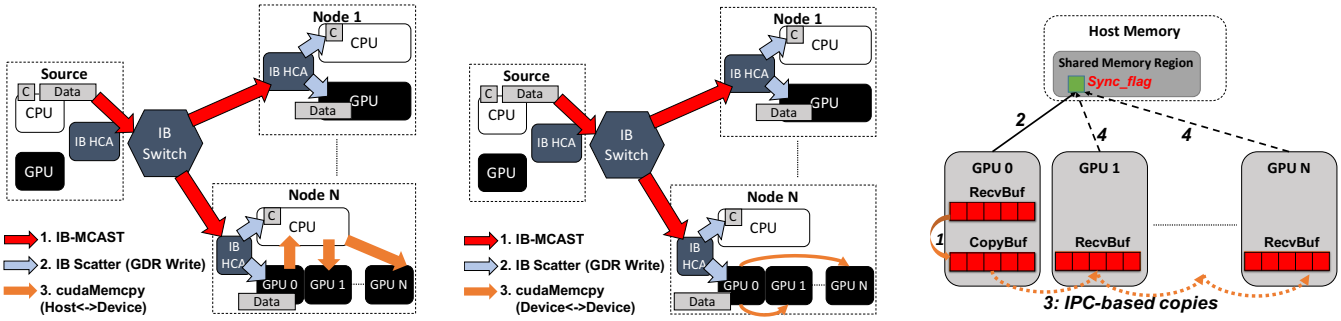
6 Designing Topology-aware Broadcast for Multi-GPU Systems

To maximize the benefits of GPUs, emerging GPU cluster systems have *dense* GPU configurations, where each node includes several GPUs connected via PCIe and/or NVLink, and with potentially varying topologies. Although variants of the proposed broadcast designs can be used for such multi-GPU systems, the participation of multiple processes per node in the inter-node communication leads to performance degradation and scalability issues [18], [34]. HCA communications bottlenecks account for some of the degradation, for an HCA may serve multiple GPUs. Indeed, in such situations, an HCA may be responsible for handling different copies of the same broadcast packet. Furthermore, in this case, the IB HCA is limited to sequentially performing writes to GPU memory in the order of the arrival of separate messages. Hence, broadcast performance may not be optimized in such multi-GPU scenarios. To address this deficiency, we explore a topology-aware broadcast mechanism in multi-GPU systems.

State-of-the-art collective algorithms generally use bi-level communication schemes. At the first level of communication, each node selects a leader process to perform the inter-node operation. Then at the second level, the leader process on each node performs intra-node broadcast to all other recipients in the same physical node. These implementations can be simply extended to GPU systems. In this case, the intra-node broadcast at the second level of communication will have the following two steps: 1) a host-based shared memory exchange, where the data is first copied to the shared memory region in the host memory, and 2) each process copies the data from shared memory on the host to its buffer in GPU memory. As shown in Figure 4(a), this approach involves expensive data movements between host and GPU memory. Further, this also consumes PCIe resources between host and device, which may limit the overall performance of streaming applications, e.g., when a streaming application would otherwise employ PCIe links for other communications.

To keep precious PCIe resources free for applications and to design efficient broadcast operation for multi-GPU systems, we have extended the SL-based design with a bi-level approach and an intra-node topology-aware scheme as shown in Figure 4(b). The proposed topology-aware design needs to account for characteristics of the system, such as, 1) inter-connectivity and placement of the different GPUs in the PCIe network and 2) placement of the HCAs and their connectivity to the GPUs on the same node. Further, to avoid the well-known P2P bottleneck in current architectures as mentioned in Section 3.2, emerging multi-GPU systems may have multiple HCAs per node, e.g., one HCA per socket. The proposed design accounts for all of these system characteristics and uses one leader process per HCA (or socket) rather than one per node. The motivation behind this design includes 1) avoiding the P2P bottleneck by ensuring that GPUs and IB devices do not communicate across sockets and 2) maximizing the benefits of the MCAST support at the first inter-node level using the proposed SL-based design.

Once leader processes receive the data of a broadcast operation on their GPUs using the SL-design, all the processes on the same socket directly read the data from GPU memory of the leader to their respective GPU memories in parallel using the CUDA IPC feature, as mentioned in Section 2.2. In order for a remote process to gain access a local GPU memory, IPC memory handlers need to be exchanged and opened beforehand between the pair of processes, which is known to have significant overhead [5]. To avoid such overhead in the critical path, in the initialization phase, each process exposes a memory region and creates an IPC handler for it. To avoid the need for an extra all-to-all exchange of these handlers inside each group, a table is established in a shared memory region for storing IPC memory handlers information. Thus, processes can directly access memory spaces of other GPU devices without additional overhead during the performance-critical broadcast operation. It is worth noting that the reason behind the all-to-all kind of pattern exchange is to provide a generic platform for designing topology-aware algorithms. For example, one can



(a) Existing bi-level shared-memory-based broadcast scheme

(b) Proposed IPC-based bi-level broadcast scheme

(c) Specific steps of the proposed IPC-based intra-node broadcast scheme

Fig. 4. Overview of the existing and proposed broadcast schemes for multi-GPU systems

design the broadcast as a parallel read from the leader with reads, writes, or mixed reads and writes, such as ring, K-nomial, or other similar types of approaches for the intra-node broadcast operation. It is also worth noting that the proposed IPC-based intra-node design is generic and applicable also to NVIDIA NVLink systems. NVLink interconnect technology is expected to offer improved performance of IPC communication (load/store operations) with higher bandwidth for the proposed parallel "Get" operations [20].

The specific steps of the proposed design are illustrated in Figure 4(c). Data transfer operations are performed directly between GPU and HCA memories, bypassing host memory. Coordination and synchronization between different processes and their leaders are accomplished using *Sync_flags* in host shared memory. To maximize pipelined overlap when using IPC schemes and asynchronicity between a leader and non-leaders, we utilize a circular chunk-based scheme. Once the leader receives data for one chunk, it copies the data into a circular buffer in GPU memory, *CopyBuf*, then it notifies non-leaders by setting the corresponding *Sync_flag* in host memory and moves forward to receive the next chunk from IB-MCAST group. Once the non-leaders finish handling their respective copy operations, they update the status of the corresponding chunks in order to notify the leader. As these copy operations are performed in a pipelined manner and overlap with the inter-node IB-MCAST operation, the overhead is well-hidden.

7 Performance Evaluation

Here, we present the performance evaluation and analysis of the proposed broadcast designs. We implemented the proposed designs in the `MPI_Bcast` (MPI blocking broadcast) function on top of MVAPICH2-GDR 2.3a, which is a CUDA-aware MPI library that leverages modern GPUDirect and InfiniBand features.

Experiments were carried out on two GPU clusters: 1) a Swiss National Supercomputing Centre (**CSCS**) cluster that is a Cray CS-Storm GPU-based system, with each node consisting of two 12-core Intel (Haswell) Xeon E5-2690 v3 2.6 GHz processors, 256 GB DDR4 memory, eight NVIDIA Tesla K80 GPU cards and two FDR IB HCAs, allowing for evaluation of the intra-node designs in conjunction with the inter-node designs. We were allowed to experiment on the CSCS system with up to 88 GPUs across 11 nodes. 2) **RI2** cluster at The Ohio State University. The system comprises 20 nodes connected via Mellanox SB7790 and SB7800 InfiniBand switches, each node equipped with two 14-core Intel (Broadwell) Xeon E5-2680 v4 2.4 GHz processors, 128 GB DDR3 Memory, one NVIDIA Tesla K80 GPU card, and one single port InfiniBand EDR HCA. For a fair comparison, we used only one GPU per node to stress the inter-node communication on the RI2 cluster, and 8 GPUs per node to stress a mix of inter- and intra-node communication on CSCS cluster.

We compared the performance of the proposed designs, labeled as *Zcpy-MCAST-GDR-Pipeline* and *TA-Zcpy-MCAST-GDR-Pipeline*, with the broadcast designs as discussed in Section 3.1, labeled as *Ring-GDR-Pipeline*, *Knomial-GDR*, and our previous work *MCAST-GDR* [30] and *MCAST-GDR-Pipeline* [31]. (We use this nomenclature for the remainder of the paper.) As discussed in Section 3, there are multiple variants of ring- and K-nomial-based algorithms for GPU-based broadcast operations. We evaluated experimentally all such variants and present the best-performing one.

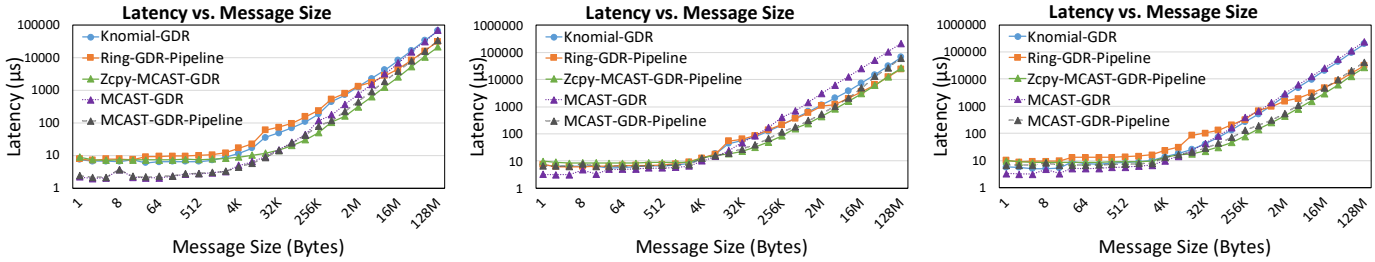
7.1 Micro-Benchmark Level Evaluation

We first present a micro-benchmark level evaluation using the OSU Micro-Benchmark (OMB) suite [6]. We utilized the *osu_bcast* benchmark to evaluate the performance of a single broadcast call. We also developed modified benchmarks that allow for examining the performance of heterogeneous and multi-source broadcast operations, respectively. Our experimental results are averaged over 5,000 iterations.

7.1.1 Single-source Heterogeneous Broadcast

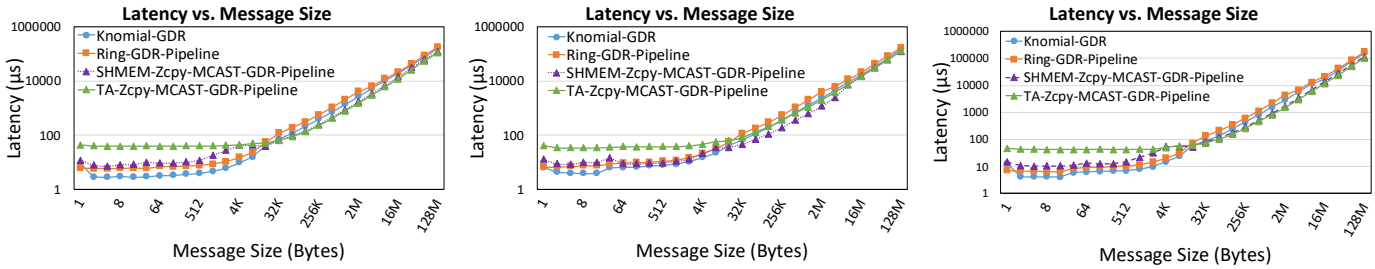
In order to simulate heterogeneous broadcast operations, we modified the current OMB suite to enable the heterogeneous configuration where the root process allocates source data in host memory with other processes expecting to receive data in GPU memory, and vice versa. While both host-to-GPU and GPU-to-host broadcast scenarios were examined for increased understanding, we note that GPU-to-host broadcast operations are not commonly used in applications.

For the broadcast operations of large messages, the proposed zero-copy and streaming-based transfer significantly improve the overall performance. In pure inter-node broadcast scenarios, as can be seen in Figures 5(a) and 5(b), the proposed design *Zcpy-MCAST-GDR-Pipeline* yields up to 51%, 82%, 88%, and 56% lower latency than the *Knomial-GDR*, *Ring-GDR-Pipeline*, *MCAST-GDR*, and *MCAST-GDR-Pipeline* designs, respectively. However, the proposed zero-copy scheme has extra synchronization overhead as described in Section 4.2; this overhead is significant for small messages, as can be observed in Figures 5(a) and 5(b). In the mix of inter- and intra-node broadcast operation, we compared the proposed topology-aware IPC-based design (*TA-Zcpy-MCAST-GDR-Pipeline*) to the existing shared memory-



(a) Host-to-GPU Broadcast (b) GPU-to-Host Broadcast (c) GPU-to-GPU Broadcast

Fig. 5. Latency comparison of single-source broadcast scenarios across 16 GPU nodes on RI2 cluster



(a) Host-to-GPU Broadcast (b) GPU-to-Host Broadcast (c) GPU-to-GPU Broadcast

Fig. 6. Latency comparison of single-source broadcast across 88 GPUs on GPU-dense CSCS cluster

based intra-node broadcast with the proposed MCAST solution (*SHMEM-Zcpy-MCAST-GDR-Pipeline*), as well as the flat K-nomial- and Ring-based designs. As shown in Figures 6(a) and 6(b), the proposed *TA-Zcpy-MCAST-GDR-Pipeline* design yields up to 44%, 63%, and 18% lower latency than the *Knomial-GDR*, *Ring-GDR-Pipeline*, and *SHMEM-Zcpy-MCAST-GDR-Pipeline* schemes, respectively, for large messages. Existing K-nomial- and Ring-based schemes take advantage of the advanced designs using point-to-point primitives in MVAPICH2-GDR to yield low latency for small messages [33]. However, these approaches are undesirable for streaming applications since they consume PCIe resources between a host and GPUs, as discussed further below in Section 7.2.

7.1.2 Single-source GPU-based Broadcast

For evaluating single-source (root) MPI_Bcast operations, we used the OMB *osu_bcast* test to compare the performance of the proposed design with existing designs. The results for a single broadcast scenario across 16 GPU nodes is shown in Figure 5(c); the proposed design shows significant performance improvement. One can observe up to 86%, 79%, 88%, and 40% reduction in latency with the proposed design compared to *Knomial-GDR*, *Ring-GDR-Pipeline*, *MCAST-GDR*, and *MCAST-GDR-Pipeline*, respectively, for message sizes ranging from 4 KB to 128 MB, which encompasses typical message sizes used at large scale in DL frameworks as illustrated in Figure 1(a). In this range of sizes, scalability is an issue for the *Knomial-GDR* and *Ring-GDR-Pipeline* cases, and the advanced IB-MCAST designs (*MCAST-GDR-Pipeline* and *Zcpy-MCAST-GDR-Pipeline*) outperform those cases. The proposed design with streaming and pipelined features avoids the P2P read limitation; this mainly why it outperforms the existing IB-MCAST designs for messages larger than 512 KB. Similarly, in a GPU-dense system, we observe up to 48%, 64%, and 16% reduction in latency of the proposed design compared to *Knomial-GDR*, *Ring-GDR-Pipeline*, and *SHMEM-MCAST-GDR-Pipeline*, respectively, for message sizes ranging from 4 KB to 128 MB, as shown in Figure 6(c). Here, we can see the proposed CUDA IPC-based intra-node broadcast brings significantly overhead for small messages.

Next, we examined scalability of the broadcast designs. As shown in Figure 7, we can see the latency of the *Ring-GDR-Pipeline* case growing linearly with increasing numbers of GPU nodes, though this case does have the lowest latency found at the small scale. As expected from our model presented in Section 3.2, the *Knomial-GDR* case is a near logarithmic. Finally, the proposed design (*Zcpy-MCAST-GDR-Pipeline*) employs IB-MCAST and exhibits stable latency independent of the number of GPU nodes.

7.1.3 Multi-Source GPU-based Broadcast

To explore multi-source broadcast scenarios, we present the most commonly used all-to-all (or an all source) broadcast variant in Figure 8. Essentially, each rank involved in the communication calls `MPI_Bcast` with itself being the source of the broadcast. This modified benchmark models the underlying communications kernel of deep learning frameworks and is discussed subsequently in Section 7.3. Figure 8 depicts performance improvements with up to 87% and 78% reduced latency of the proposed design over *Knomial-GDR* and *Ring-GDR-Pipeline* approaches, respectively. Like in the single-source broadcast scenario, *Knomial-GDR* and *Ring-GDR-Pipeline* schemes suffer from the P2P read limitation scalability issue.

For the all-to-all broadcast scenario, the number of broadcast sources doubles commensurate with doubling the number of GPU nodes. Therefore, the best-case scenario is that latency increases linearly with increased network scale. In Figure 9, we observe the proposed design having near linear latency growth with increasing scale, outperforming the existing broadcast schemes (beginning at four nodes).

7.2 Evaluating Streaming Workloads

We developed a synthetic streaming benchmark to mimic the behavior of streaming applications. In the synthetic

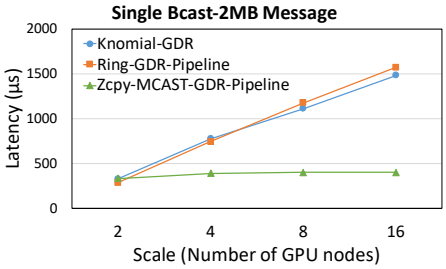


Fig. 7. Scalability analysis for single-source broadcast

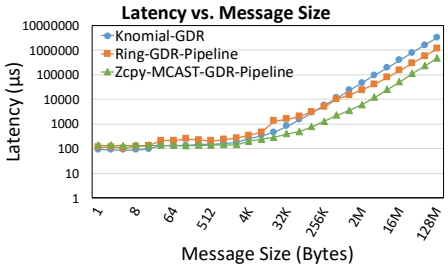


Fig. 8. Performance comparison for all-to-all broadcast on 16 GPUs

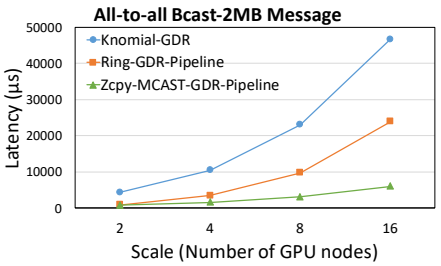


Fig. 9. Scalability analysis of All-to-all Broadcast

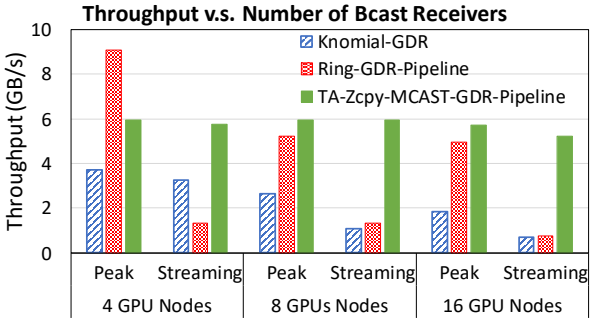


Fig. 10. Performance comparison of the synthetic streaming benchmark across up to 16 GPU nodes

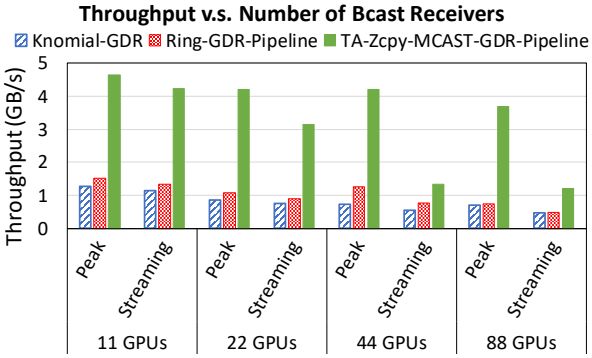


Fig. 11. Performance comparison of the synthetic streaming benchmark on a GPU-dense cluster with up to 88 GPUs across 11 nodes

streaming benchmark, using *cudaMemcpyAsync* we kept injecting background data traffic from host to GPU memory for the broadcast source, and data from GPU to host memory for the broadcast receivers. Meanwhile, a window of back-to-back `MPI_Bcast` operations is performed. This synthetic streaming benchmark simulates competition for PCIe resources, which typifies streaming applications. Here, we compared the proposed solution (*TA-Zcpy-MCAST-GDR-Pipeline*) to *Knomial-GDR* and *Ring-GDR-Pipeline* schemes. Towards estimating the impact of the various schemes for streaming applications, we measured throughput of the benchmark on RI2 and CSCS systems.

To observe the tolerance of background PCIe traffic in these broadcast schemes, we present the 'Peak' throughput a scheme can achieve without background streaming traffic and the throughput when 'Streaming' begins. As shown in Figure 7.2, the proposed design (*IPC TA-Zcpy-MCAST-GDR-Pipeline*) achieves **stable throughput**, which is around 6 GB/s, independent of the number of GPU nodes, which is essential for streaming processes. Although the Ring-based design achieves the highest peak throughput (near 9 GB/s) in the small scale, Ring- and K-nomial-based designs do not scale well and have significantly negative impact on streaming traffic.

These results are consistent with the desired objective of increasing the efficiency of PCIe resource utilization in implementing broadcast operations towards maximizing the availability of such resources for application use.

Next, we conducted the same experiments on the GPU-dense CSCS system. As shown in Figure 7.2, the proposed designs provide significantly higher throughput than K-nomial- and Ring-based schemes. Nevertheless, there is unavoidable PCIe-level contention between background traffic and the proposed IPC-based intra-node broadcast operations, which involve reading data from the same GPU. It is worth noting that such contention may not be exist in NVLink-enabled systems, which employ direct GPU-to-GPU NVLink connections [20]. Therefore, the proposed designs utilize IPC-based device-to-device data transfer operations, freeing host-to-device/device-to-host PCIe channels, availing such data paths for applications. Collectively, the results suggest the proposed designs can maximize MPI broadcast throughput and reduce processing time for streaming applications.

7.3 Evaluating Deep Learning Workloads

To perform an application-level evaluation we incorporate what we learned empirically from the benchmark-level evaluation of the previous section. Here, we compare the proposed streaming broadcast design to the *Ring-GDR-Pipeline* and *Knomial-GDR* approaches. From Section 7.1, we know that the proposed IB-MCAST-based design provides significant performance improvement for messages larger than 128 KB and that the existing *MCAST-GDR* design performs well for 4 to 128 KB messages since those sizes do not encounter the P2P read limit. Thus, we have used the existing IB-MCAST-based *MCAST-GDR* for messages up to 128 KB and have applied the proposed design *TA-Zcpy-MCAST-GDR-Pipeline* for messages beyond this in the application level evaluation.

The Microsoft Cognitive Toolkit (CNTK) is a recently introduced and widely-used DL toolkit [12]. Here, we focus on CNTK data parallel stochastic gradient descent (SGD) training that utilizes an all-to-all broadcast pattern during its gradient aggregation phase. For our evaluation we have utilized an optimized version of the CNTK code [13]. The experiments were conducted on the RI2 cluster to perform image classification training using images larger than 32 GB from the ImageNet dataset [14].

7.3.1 AlexNet

One of the most popular and heavily cited DL networks is the AlexNet network [38]. Figure 12 highlights a compar-

ison of K-nomial-GDR, Ring-GDR-Pipeline, and TA-Zcpy-MCAST-GDR-Pipeline designs for the average training time in one epoch. We present results utilizing 8- and 16-GPU data-parallel training cases with batch size 128. This corresponds to using maximum buffer sizes of approximately 18 and 9 MB in the 8- and 16-GPU cases, respectively, for broadcast operations. The broadcast primitive is called throughout the training process with varying message sizes ranging from 48 bytes to around 18 MB. It is encouraging to note that while the broadcast time is just part of the gradient aggregation process as indicated in Figure 1(a), we have been able to reduce the overall training time by 24% and 15% using the proposed streaming-based IB-MCAST design compared to K-nomial-GDR and Ring-GDR-Pipeline designs, respectively, over 16 GPUs, notably with no application-level code changes.

7.3.2 VGG

VGG is another important DL network that utilizes a larger model size, i.e., deeper networks [39]. For example, the size of the communication buffer is up to about 200 MB for two processes across two GPUs. In these experiments, due to the memory limitation of GPU, the batch size used is 32, which results in maximum buffer sizes of about 50 and 25 MB with 8 and 16 GPU nodes, respectively. Figure 12 compares training performance for both 8 and 16 GPUs using *Knomial-GDR*, *Ring-GDR-Pipeline*, and *TA-Zcpy-MCAST-GDR-Pipeline* designs. The proposed design (*TA-Zcpy-MCAST-GDR-Pipeline*) reduces by 16% and 7% the overall training time compared to K-nomial-GDR and Ring-GDR-Pipeline designs, respectively, on 16 GPUs. Moreover, we observe performance improvements with increasing scale. This indicates that further benefits may be expected from the proposed design for GPU clusters larger than those to which we had access.

7.3.3 ResNet-50

Residual network (ResNet) is a super-deep neural network for residual learning [40]. In employing super-deep layers, the size of data exchanged via MPI is small compared to that in AlexNet and VGG networks. In our experiments, the batch size is set to 32, and it results in using up to about 1.1 MB and 578 KB broadcast buffers in the 8- and 16-GPU cases, respectively. Furthermore, the majority of message sizes for broadcast operations are in the range 4 to 64 KB. Nothwithstanding that such sizes do not fully take advantage of the proposed pipelined MCAST design, the proposed design still reduces by 5% and 18% the training time against K-nomial-GDR and Ring-GDR-Pipeline designs, respectively, on 16 GPUs, as shown in Figure 12. With increasing scale we anticipate further performance benefits of the proposed MCAST design.

8 Model Validation and Prediction

In this section, based on the performance results presented in Section 7, we evaluate the proposed models of sections 3 and 5. We also predict broadcast performance in multi-source broadcast scenarios on large-scale GPU systems.

For experiments on the RI2 cluster, we measured the required parameters (Table 1) for the proposed models, finding $C = 512$ KB, $U = 4$ KB, $B_H \approx 100$ Gbps, and $B_{H2D} \approx 8$ Gbps. Note that t_s is negligibly small

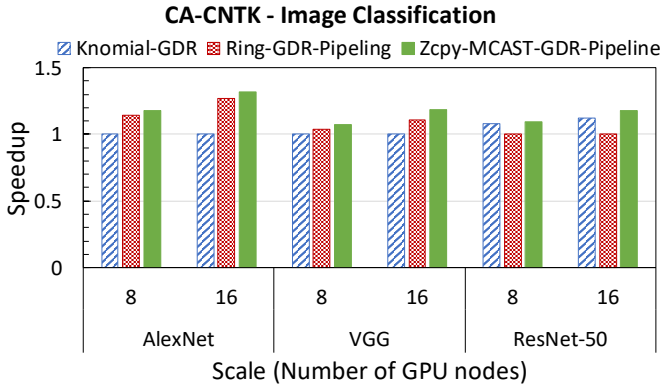


Fig. 12. Speedup of the proposed designs compared to existing schemes for DL networks across 16 GPU nodes

and can be ignored without loss of generality. Since the overhead, $t_o(n)$, of the IB-MCAST operation itself has not been thoroughly studied in the literature, we developed a model with the reasonable assumption that the number of nodes involved impacts IB-MCAST overhead. Based on experimental results, we used least squares-based curve fitting to approximate it as a logarithmic function.

Our approximation shows the overhead of IB-MCAST can be formulated as follows:

$$t_o(n) \approx \frac{1}{a} \times \ln(n)\,,\ 15 \leq a \leq 20 \tag{10}$$

As given in Equation 10, compared to the cost of ring-based algorithms, which is approximately $O(n)$, we readily see that IB-MCAST-based schemes, i.e., Equations 7, 8, and 9, have significantly lower communication cost for large-scale clusters. For K-nomial-based algorithms, the communication cost is mainly dominated by $\lceil \log_k n \rceil$, which can be transformed to $\lceil \frac{1}{\ln(k)} \times \ln(n) \rceil$. This means that K-nomial-based algorithms may only have theoretically lower overhead compared to IB-MCAST-based schemes for thousands to millions of GPU nodes, i.e., when $\ln(k) > a$. In practice, however, K-nomial-based schemes are implemented with multiple point-to-point operations, which cannot be issued simultaneously as the theoretical models expect when $k = n$. In practice, on typically sized GPU-enabled HPC systems [41], then, IB-MCAST-based schemes are expected to have less overhead than K-nomial-based schemes.

For the evaluation, we have selected a 2 MB message size, which is the most commonly used size for broadcast operations in our application-level evaluation with CNTK as presented in Section 7.3. First, we can calculate the fractional overlap (OP) in a multi-source broadcast scenario based on the latencies of same-sized single- and multi-source broadcast operations (T_{Single_Bcast} and T_{Multi_Bcast}, respectively) and on the number of broadcast sources (m). This can be represented as follows:

$$OP = \frac{m * T_{Single\_Bcast} - T_{Multi\_Bcast}}{m * T_{Single\_Bcast}} \tag{11}$$

Second, based on results of Section 7, we can determine that the fractional overlaps (as percentages) in the proposed design for 2 MB messages is 4%, 12%, 13%, and 20% for 2, 4, 8, and 16 GPU nodes, respectively. Similarly, in the multi-source broadcast scenario, we can measure the overlap for K-nomial-based and ring-based schemes. We discovered

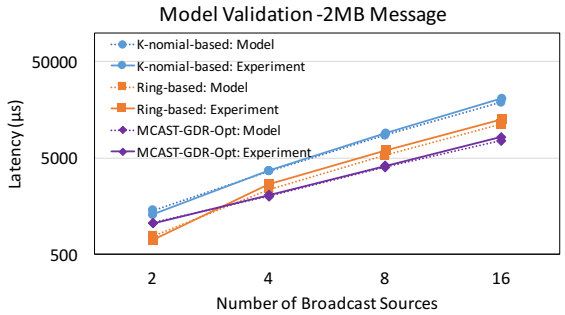


Fig. 13. Model validation for multi-source broadcast on GPU clusters

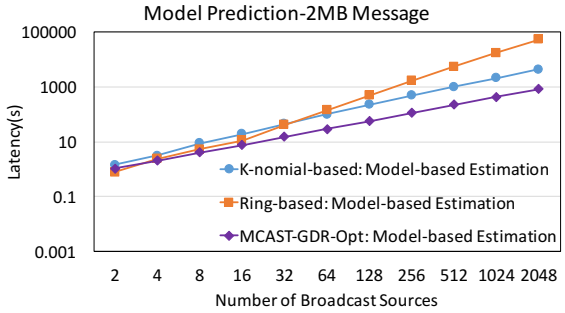


Fig. 14. Performance prediction of multi-source broadcast for large-scale GPU clusters

that K-nomial-based schemes have nearly no overlap and that a pipelined ring-based scheme can achieve about 50% overlap with 16 broadcast sources. However, ring-based schemes, with their scalability deficiencies, have lower performance compared to the proposed schemes. It is worth noting that since K-nomial-based and ring-based schemes are implemented via multiple non-blocking point-to-point communication patterns, i.e., like MPI_Isend/MPI_Irecv, in the MVAPICH2 library, there is a certain amount of overlap between these calls within a single-source broadcast. We account for this when validating the models and making performance predictions by including a 20% overlap in the single-source broadcast models of Section 3.

Next, we can enter these parameters into our models as shown in Section 3 to evaluate the models. Figure 13 compares the model-based performance estimation to experimental results as reported in the previous section. Theoretical and experimental performance being within 10% of each other validates the accuracy of the proposed performance model. Finally, using our performance estimation models, we can predict the performance of the various designs on a larger GPU cluster. To simplify predictions, we assume all schemes gain a 5% additional overlap with each doubling of the number of broadcast sources. As shown in Figure 14, the proposed scheme has the lowest latency, which is due to it exploiting the highly scalable IB-MCAST feature. Although ring-based schemes achieve the greatest level of overlap, the high latency of inter-node data transfers dominates performance.

9 Related Work

This section discusses existing studies that exploit hardware and software features of InfiniBand HCAs and GPUs for communications in GPU clusters.

Solely host-based MPI designs for exploiting IB-MCAST appear in [18], [34]. These studies were first to highlight the potential of using IB-MCAST in designing efficient broadcast primitives. Using simulators, Zhou et al. [16] proposed a cyclic multicast scheme to improve the performance of multicast operations in fat-tree-based IB networks. Hoefler et al. [17] presented a host-based broadcast implementation that utilizes hardware multicast features with a ring-based algorithm and allows for a near constant-time complexity and included but a micro-benchmark level evaluation.

Venkatesh et al. [19] present homogeneous support for utilizing the IB-MCAST feature in tandem with GPUDirect RDMA. Potluri et al. [5] leveraged CUDA IPC to improve the programmability as well as the performance of intra-node communication on multi-GPU systems. For such systems, various studies have been published that include optimization of MPI runtimes [5], [42], benchmarks [4], [43], [44], and applications [1], [3]. Sourouri et al. [42] proposed overlapping computation and communication through using a combination of multiple CUDA streams and OpenMP threads. Our earlier work [30] provides preliminary support of IB-MCAST-based designs for heterogeneous broadcast as presented in sections 4.1 and 6 on multi-GPU systems. In [31], the focus was on GPU-resident data for deep learning workloads as explained in Section 5. In the present paper, we further discuss improved broadcast schemes that provide efficient performance for various combinations of communication patterns.

10 Concluding Remarks

Deep learning applications typically employ broadcast operations with large messages for distributing workloads to GPU computing sites in HPC clusters. In this paper, through an analytical modeling approach, we observed multiple fundamental bottlenecks in existing broadcast schemes for transferring from multiple sources medium and large messages among GPUs in an InfiniBand network. Based on this analysis and by exploiting advanced features of modern HPC cluster components, such as InfiniBand hardware multicast and GPUDirect RDMA, we proposed an efficient and scalable GPU-based multi-source streaming broadcast design. Through a detailed performance evaluation and analysis of the design and models on a real-world GPU-enabled InfiniBand cluster, using streaming benchmarks on 88 GPUs across 11 nodes we observed stable and higher throughput from the proposed designs as well as up to 64% reduction in latency compared to existing broadcast schemes. Furthermore, a performance evaluation of the modified CUDA-aware deep learning toolkit based on Microsoft CNTK across 16 GPU nodes found the proposed design yields 15%, 7%, and 18% reduction in training time compared to the existing ring-based pipeling scheme for AlexNet, VGG and ResNet networks, respectively, without loss of accuracy and notably with no application changes.

Future plans include extending the proposed designs to other broadcast-based collective operations, such as all-reduce, all-gather, and all-to-all, as well as evaluating the proposed designs with various streaming and deep learning applications in upcoming large-scale GPU-dense clusters.

Acknowledgments

This material is based upon work supported by, or in part by, the United States Department of Defense (DOD) High

Performance Computing Modernization Program (HPCMP) User Productivity Enhancement and Technology Transfer (PETTT) contract #GS04T09DBC0017. Any opinions, findings, and conclusions or recommendations expressed herein are those of the authors and do not necessarily reflect the views of the DOD HPCMP or the employer of the author.

We thank Drs. Sadaf Alam and Carlos Osuna for providing access to the Swiss National Supercomputing Centre (CSCS) testbed.

REFERENCES

[1] A. Vilches, A. Navarro, R. Asenjo, F. Corbera, R. Gran, and M. J. M. J. Garzarán, "Mapping Streaming Applications on Commodity Multi-CPU and GPU On-Chip Processors," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 4, pp. 1099–1115, April 2016.

[2] A. A. Awan, K. Hamidouche, J. M. Hashmi, and D. K. Panda, "S-Caffe: Co-designing MPI Runtimes and Caffe for Scalable Deep Learning on Modern GPU Clusters," in *Proceedings of the 22Nd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, ser. PPoPP '17. ACM, 2017, pp. 193–205.

[3] H. T. Meng and J. M. Jin, "Acceleration of the Dual-Field Domain Decomposition Algorithm Using MPI-CUDA on Large-Scale Computing Systems," *IEEE Transactions on Antennas and Propagation*, vol. 62, no. 9, pp. 4706–4715, Sept 2014.

[4] G. Jo, J. Nah, J. Lee, J. Kim, and J. Lee, "Accelerating LINPACK with MPI-OpenCL on Clusters of Multi-GPU Nodes," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 7, pp. 1814–1825, July 2015.

[5] S. Potluri, H. Wang, D. Bureddy, A. K. Singh, C. Rosales, and D. K. Panda, "Optimizing MPI Communication on Multi-GPU Systems Using CUDA Inter-Process Communication," in *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2012 IEEE 26th International*, May 2012, pp. 1848–1857.

[6] D. Bureddy, H. Wang, A. Venkatesh, S. Potluri, and D. K. Panda, "OMB-GPU: A Micro-benchmark Suite for Evaluating MPI Libraries on GPU Clusters," in *Proceedings of the 19th European Conference on Recent Advances in the Message Passing Interface (EuroMPI)*, 2012, pp. 110–120.

[7] B. Agarwalla, N. Ahmed, D. Hilley, and U. Ramachandran, "Streamline: A Scheduling Heuristic for Streaming Applications on the Grid," in *Electronic Imaging 2006*. International Society for Optics and Photonics, 2006, pp. 607 107–607 107.

[8] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional Architecture for Fast Feature Embedding," *arXiv preprint arXiv:1408.5093*, 2014.

[9] NVIDIA, "Caffe: a fast open framework for deep learning," Accessed: July 25, 2018. [Online]. Available: https://github.com/NVIDIA/caffe

[10] Facebook, "A New Lightweight, Modular, and Scalable Deep Learning Framework," Accessed: July 25, 2018. [Online]. Available: https://caffe2.ai/

[11] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado *et al.*, "TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems," Accessed: July 25, 2018. [Online]. Available: http://tensorflow.org/

[12] Microsoft, "The Microsoft Cognitive Toolkit," Accessed: July 25, 2018. [Online]. Available: http://www.cntk.ai/

[13] D. S. Banerjee, K. Hamidouche, and D. K. Panda, "Re-Designing CNTK Deep Learning Framework on Modern GPU Enabled Clusters," in *2016 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, Dec 2016, pp. 144–151.

[14] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[15] "InfiniBand Trade Association," Accessed: July 25, 2018. [Online]. Available: http://www.infinibandta.org

[16] J. Zhou, X.-Y. Lin, and Y.-C. Chung, "Hardware Supported Multicast in Fat-tree-based InfiniBand Networks," *J. Supercomput.*, vol. 40, no. 3, pp. 333–352, Jun. 2007.

[17] T. Hoefler, C. Siebert, and W. Rehm, "A Practically Constant-time MPI Broadcast Algorithm for Large-scale InfiniBand Clusters with Multicast," in *Proceedings of the 21st IEEE International Parallel & Distributed Processing Symposium (CAC'07 Workshop)*, Mar. 2007, p. 232.

[18] A. R. Mamidala, L. Chai, H.-W. Jin, and D. K. Panda, "Efficient SMP-aware MPI-level Broadcast over InfiniBand's Hardware Multicast," in *Proceedings 20th IEEE International Parallel Distributed Processing Symposium*, April 2006, p. 8.

[19] A. Venkatesh, H. Subramoni, K. Hamidouche, and D. K. Panda, "A High Performance Broadcast Design with Hardware Multicast and GPUDirect RDMA for Streaming Applications on Infiniband Clusters," in *2014 21st International Conference on High Performance Computing (HiPC)*, Dec 2014, pp. 1–10.

[20] "NVIDIA NVLINK HIGH-SPEED INTERCONNECT," Accessed: July 25, 2018. [Online]. Available: http://www.nvidia.com/object/nvlink.html

[21] "NVIDIA GPUDirect," Accessed: July 25, 2018. [Online]. Available: https://developer.nvidia.com/gpudirect

[22] D. M. Wadsworth and Z. Chen, "Performance of MPI Broadcast Algorithms," in *2008 IEEE International Symposium on Parallel and Distributed Processing*, April 2008, pp. 1–7.

[23] J. Pjesivac-Grbovic, T. Angskun, G. Bosilca, G. E. Fagg, E. Gabriel, and J. J. Dongarra, "Performance Analysis of MPI Collective Operations," in *19th IEEE International Parallel and Distributed Processing Symposium*, April 2005, p. 8.

[24] K. Hasanov, J.-N. Quintin, and A. Lastovetsky, "High-level topology-oblivious optimization of MPI broadcast algorithms on extreme-scale platforms," in *European Conference on Parallel Processing*. Springer, 2014, pp. 412–424.

[25] T. Chiba, T. Endo, and S. Matsuoka, "High-Performance MPI Broadcast Algorithm for Grid Environments Utilizing Multi-lane NICs," in *Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGrid '07)*, May 2007, pp. 487–494.

[26] H. Zhou, V. Marjanovic, C. Niethammer, and J. Gracia, "A Bandwidth-Saving Optimization for MPI Broadcast Collective Operation," in *2015 44th International Conference on Parallel Processing Workshops*, Sept 2015, pp. 111–118.

[27] NVIDIA, "Optimized Primitives for Collective Multi-GPU Communication," Accessed: July 25, 2018. [Online]. Available: https://github.com/NVIDIA/nccl

[28] A. A. Awan, K. Hamidouche, A. Venkatesh, and D. K. Panda, "Efficient Large Message Broadcast Using NCCL and CUDA-Aware MPI for Deep Learning," in *Proceedings of the 23rd European MPI Users' Group Meeting*, ser. EuroMPI 2016. New York, NY, USA: ACM, 2016, pp. 15–22.

[29] A. A. Awan, C. Chu, H. Subramoni, and D. K. Panda, "Optimized Broadcast for Deep Learning Workloads on Dense-GPU InfiniBand Clusters: MPI or NCCL?" *arXiv preprint arXiv:1707/09414*, vol. abs/1707.09414, 2017.

[30] C.-H. Chu, K. Hamidouche, H. Subramoni, A. Venkatesh, B. Elton, and D. K. Panda, "Designing High Performance Heterogeneous Broadcast for Streaming Applications on GPU Clusters," in *2016 28th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, Oct 2016, pp. 59–66.

[31] C.-H. Chu, X. Lu, A. A. Awan, H. Subramoni, J. Hashmi, B. Elton, and D. K. Panda, "Efficient and Scalable Multi-Source Streaming Broadcast on GPU Clusters for Deep Learning," in *46th International Conference on Parallel Processing (ICPP-2017)*, Aug 2017.

[32] S. Potluri, K. Hamidouche, A. Venkatesh, D. Bureddy, and D. Panda, "Efficient Inter-node MPI Communication Using GPUDirect RDMA for InfiniBand Clusters with NVIDIA GPUs," in *Parallel Processing (ICPP), 2013 42nd International Conference on*, Oct 2013, pp. 80–89.

[33] R. Shi, S. Potluri, K. Hamidouche, J. Perkins, M. Li, D. Rossetti, and D. K. Panda, "Designing Efficient Small Message Transfer Mechanism for Inter-node MPI Communication on InfiniBand GPU Clusters," in *2014 21st International Conference on High Performance Computing (HiPC)*, Dec 2014, pp. 1–10.

[34] J. Liu, A. R. Mamidala, and D. K. Panda, "Fast and Scalable MPI-level Broadcast using InfiniBand's Hardware Multicast Support," in *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, April 2004, p. 10.

[35] M. Li, S. Potluri, K. Hamidouche, J. Jose, and D. K. Panda, "Efficient and Truly Passive MPI-3 RMA Using InfiniBand Atomics," in *Proceedings of the 20th European MPI Users' Group Meeting*, ser. EuroMPI '13. New York, NY, USA: ACM, 2013, pp. 91–96.

[36] R. Gerstenberger, M. Besta, and T. Hoefler, "Enabling Highly-Scalable Remote Memory Access Programming with MPI-3 One Sided," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. ACM, 11 2013, pp. 53:1–53:12.

[37] C. H. Chu, K. Hamidouche, H. Subramoni, A. Venkatesh, B. Elton, and D. K. Panda, "Efficient Reliability Support for Hardware Multicast-Based Broadcast in GPU-enabled Streaming Applications," in *2016 First International Workshop on Communication Optimizations in HPC (COMHPC)*, Nov 2016, pp. 29–38.

[38] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[39] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[40] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016, pp. 770–778.

[41] "Top 500 Supercomputer sites," Accessed: July 25, 2018. [Online]. Available: http://www.top500.org/

[42] M. Sourouri, T. Gillberg, S. B. Baden, and X. Cai, "Effective Multi-GPU Communication using Multiple CUDA Streams and Threads," in *2014 20th IEEE International Conference on Parallel and Distributed Systems (ICPADS)*, Dec 2014, pp. 981–986.

[43] I. B. Peng, S. Markidis, E. Laure, D. Holmes, and M. Bull, "A Data Streaming Model in MPI," in *Proceedings of the 3rd Workshop on Exascale MPI*, ser. ExaMPI '15. New York, NY, USA: ACM, 2015, pp. 2:1–2:10.

[44] S. Kamburugamuve, M. Pathirage, S. Ekanayake, and G. C. Fox, "High Performance Processing of Streaming Data," in *2015 IEEE 22nd International Conference on High Performance Computing Workshops (HiPCW)*, Dec 2015, p. 58.

Ching-Hsiang Chu received B.S. and M.S. degrees in Computer Science and Information Engineering from National Changhua University of Education, Taiwan in 2010 and National Central University, Taiwan in 2012, respectively. He is currently working towards a Ph.D. degree in Computer Science and Engineering at The Ohio State University, Columbus, Ohio. His research interests include high-performance computing and networking, wireless networks, and cloud computing. More details are available at http://web.cse.ohio-state.edu/~chu.368.

Xiaoyi Lu received a Ph.D. degree in Computer Science from the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China. He is a research scientist in the Department of Computer Science and Engineering, Ohio State University, USA. His current research interests include high performance interconnects and protocols, Big Data, Hadoop/Spark/Memcached Ecosystem, Parallel Computing Models (MPI/PGAS), Virtualization, Cloud Computing, and Deep Learning. He is currently leading the design and development for the High-Performance Big Data (HiBD) project (http://hibd.cse.ohio-state.edu). The HiBD packages are currently being used by more than 275 organizations in 34 countries. More than 25,150 downloads of these libraries have taken place. He has published more than 80 papers in major journals and international conferences related to these research areas and is actively involved in various professional activities in academic journals and conferences. He is a member of the IEEE and ACM. More details about Dr. Lu are available at http://web.cse.ohio-state.edu/~lu.932.

Ammar A. Awan received his B.S. and M.S. degrees in Computer Science and Engineering from National University of Science and Technology, Pakistan and Kyung Hee University, South Korea, respectively. Currently, Ammar is working towards his Ph.D. degree in Computer Science and Engineering at The Ohio State University. His current research focus lies at the intersection of High Performance Computing libraries and Deep Learning frameworks. He previously worked on a Java-based Message Passing Interface (MPI) and nested parallelism with OpenMP and MPI for scientific applications. More details are available at http://web.cse.ohio-state.edu/~awan.10.

Hari Subramoni is a research scientist in the Department of Computer Science and Engineering at the Ohio State University, USA, since September 2015. His current research interests include high performance interconnects and protocols, parallel computer architecture, network-based computing, exascale computing, network topology aware computing, QoS, power-aware LAN-WAN communication, fault tolerance, virtualization, big data and cloud computing. He has published over 50 papers in international journals and conferences related to these research areas. He has been actively involved in various professional activities in academic journals and conferences. Dr. Subramoni is doing research on the design and development of MVAPICH2 (High Performance MPI over InfiniBand, iWARP and RoCE) and MVAPICH2-X (Hybrid MPI and PGAS (OpenSHMEM, UPC and CAF)) software packages. He is a member of IEEE. More details about Dr. Subramoni are available from http://web.cse.ohio-state.edu/~subramoni.1/.

Bracy Elton received a B.S. Cum laude, double majoring in Mathematics and Computer Science, from Pacific Lutheran University. He received M.S. and Ph.D. degrees in Computer Science from the University of California, Davis, while a student-employee at Lawrence Livermore National Laboratory. He is a Technical Fellow and Senior Computational Scientist at Engility Corporation with a focus in high performance computing (HPC) in signal/image processing (SIP) applications and systems. He holds a position as a SIP On-Site in the US DOD High Performance Computing Modernization Program's User Productivity Enhancement, Technology Transfer and Training (DOD HPCMP PETTT) activity. He co-chairs the PETTT Emerging Computational Technologies Working Group and the Emerging Accelerator Technologies Technical Focus Area. He has 26+ years of experience in HPC and computational science across multiple disciplines, including prior positions with supercomputer manufacturers and the Ohio Supercomputer Center. He has 27 publications in respected journals and conferences. He is a member of IEEE (Senior Member), IEEE Computer Society, IEEE Signal Processing Society, SIAM, SIAM Special Interest Groups (SIAGs) on Supercomputing, on Computational Science and Engineering and on Imaging Science, ACM Special Interest Group on HPC, American Mathematical Society, and Sigma Xi.

Dhabaleswar K. Panda is a professor of computer science and engineering at Ohio State University. He has published more than 400 papers in major journals and international conferences. He and his research group members have been doing extensive research on modern networking technologies including InfiniBand, High-Speed Ethernet and RDMA over Converged Enhanced Ethernet (RoCE). The MVAPICH2 (High Performance MPI over InfiniBand, iWARP and RoCE) and MVAPICH2-X software libraries, developed by his research group (http://mvapich.cse.ohio-state.edu), are currently being used by more than 2,875 organizations worldwide (in 85 countries). This software has enabled several InfiniBand clusters to get into the latest TOP500 ranking during the last decade. More than 447,000 downloads of this software have taken place from the project's website alone. The RDMA packages for Apache Spark, Apache Hadoop, and Memcached together with OSU HiBD benchmarks from his group (http://hibd.cse.ohio-state.edu) are also publicly available. These libraries are currently being used by more than 275 organizations in 34 countries. More than 25,150 downloads of these libraries have taken place. His research has been supported by funding from the US National Science Foundation, the US Department of Energy, the US Department of Defense, and several industrial supporters, including Intel, Cisco, Cray, SUN, Mellanox, QLogic, NVIDIA, Microsoft, and NetApp. He is an IEEE fellow and a member of the ACM. More details about Prof. Panda are available at http://web.cse.ohio-state.edu/~panda.2/.