NV-Group: Link-Efficient Reduction for Distributed Deep Learning on Modern Dense GPU Systems

Ching-Hsiang Chu, Pouya Kousha, Ammar Ahmad Awan, Kawthar Shafie Khorassani
Hari Subramoni and Dhabaleswar K. (DK) Panda
The Ohio State University
Columbus, Ohio
{chu.368,kousha.2,awan.10,shafiekhorassani.1}@osu.edu
{subramon,panda}@cse.ohio-state.edu

ABSTRACT

The advanced fabrics like NVIDIA NVLink are enabling the deployment of dense Graphics Processing Unit (GPU) systems such as DGX-2 and Summit. With the wide adoption of large-scale GPU-enabled systems for distributed deep learning (DL) training, it is vital to design efficient communication such as the *Allreduce* operation to achieve near-ideal speedup at scale. In this paper, we propose a link-efficient scheme through NVLink-aware cooperative reduction kernels to significantly accelerate Allreduce operations for distributed deep learning applications. By overlapping computation and communication and maximizing utilization of all available NVLinks between CPU and GPU, as well as among GPUs, we demonstrate 1.8X performance improvement of Allreduce on 1,536 GPUs compared to state-of-the-art GPU-Aware MPI and NVIDIA NCCL libraries. Finally, we demonstrate 93.9% and 89.7% scaling efficiency (i.e., 15X and 172X speedup) for training ResNet-50 models using TensorFlow on a 16-GPU DGX-2 node and on 192-GPUs of the Summit system, respectively. To the best of our knowledge, this is the first study that achieves near-ideal scaling efficiency for distributed DL training and deals with designs tailored for cutting-edge systems like DGX-2 and Summit clusters.

CCS CONCEPTS

• **Computing methodologies** → **Parallel programming languages**; • **Computer systems organization** → **Interconnection architectures**; **Single instruction, multiple data**.

KEYWORDS

Allreduce, Deep Learning, GPU, HPC, MPI, NVLink,

ACM Reference Format:
Ching-Hsiang Chu, Pouya Kousha, Ammar Ahmad Awan, Kawthar Shafie Khorassani and Hari Subramoni and Dhabaleswar K. (DK) Panda. 2020. NV-Group: Link-Efficient Reduction for Distributed Deep Learning on Modern Dense GPU Systems . In *2020 International Conference on Supercomputing (ICS '20), June 29-July 2, 2020, Barcelona, Spain.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3392717.3392771

ICS '20, June 29-July 2, 2020, Barcelona, Spain

ACM ISBN 978-1-4503-7983-0/20/06…$15.00
https://doi.org/10.1145/3392717.3392771

1 INTRODUCTION

Graphics Processing Units (GPUs) have become ever prevalent hardware in modern high-performance computing (HPC) systems and cloud platforms due to their high-bandwidth memory and massive parallelism. A broad spectrum of applications ranging from traditional HPC applications to artificial intelligence (AI) enabled applications have been significantly accelerated by exploiting GPU platforms. As a result, more powerful HPC systems and cloud platforms are being deployed with thousands of nodes [44], equipped with cutting-edge CPU, GPU and high-speed interconnects. With the increased deployment of large and dense GPU systems, efficient data movement between GPUs within and across nodes is critical for HPC and AI applications. However, data movement among GPUs, CPUs, and network devices using traditional PCIe has significantly stymied newer applications that demand higher bandwidth. To this end, new proprietary interconnect architectures such as NVIDIA NVLink [15], AMD Infinity Fabric, and Intel Xe link are introduced to provide high-bandwidth data movement dedicated for GPUs. Figure 1 depicts the logical view of a DGX-2 system with NVIDIA Volta GPUs connected by NVLink2 and NVSwitch.

NVLinks are also empowering the latest OpenPOWER systems with unprecedented intra-node bandwidth [9, 20] for GPU-based communication. Furthermore, four out of the Top 10 supercomputers [44] including Summit (#1), Sierra (#2), ABCI (#8), and Lassen (#10) are all using the NVLink interconnect. In the emerging distributed Deep Learning (DL) training, *Allreduce*, one of the legacy collective communication patterns defined in Message Passing Interface (MPI) standard [26], becomes the major bottleneck of GPU communication at scale [3–5, 7, 14]. As a result, many solutions have been proposed to address the inefficiency of Allreduce with a large amount of data on the emerging architectures [2, 6, 10–12, 34, 45, 47, 49]. However, the state-of-the-art Allreduce algorithms on the emerging dense-GPU systems are still sub-optimal,

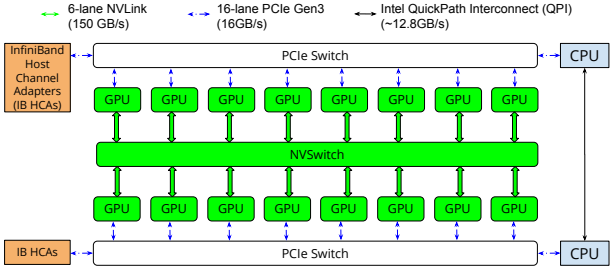


Figure 1: Hardware configuration of NVIDIA DGX2 system with cutting-edge GPU and interconnect architectures

especially for very large message sizes used frequently in the distributed deep learning training process [5]. Furthermore, link efficiency of Allreduce schemes on the emerging dense-GPU systems and its impact on performance have not been addressed in literature.

1.1 Motivation and Challenges

Achieving high-performance large-size Allreduce communication on NVLink-enabled systems requires the designer of the communication library to account for several novel architectural features available on these systems. **1) Direct LOAD/STORE capability**: Previous MPI-based solutions were performing the GPU-GPU reduction in a *COPY-COMPUTE-COPY* approach [3, 12, 24, 34], which requires extra copies and memory allocations. As the new NVLink-like architecture provides low-latency LOAD/STORE primitives between GPUs, one can utilize this capability to perform GPU-GPU reduction in an efficient *LOAD-COMPUTE-STORE* manner. **2) Maximize utilization of NVLinks between CPUs and GPUs:** When performing communication at scale, data has to be moved between CPU and GPU for inter-node communication [38]. Leveraging LOAD/STORE capability on the NVLinks between CPU and GPU can significantly improve the overall performance and load balance. **3) Overlap Reduction with communication:** Allreduce algorithm for large data sizes typically relies on multi-stage pipeline [37, 39, 49]. Therefore, it is important to achieve high overlap between compute intensive portions of the reduction operations and communication. Table 1 summarizes these properties for the state-of-the-art GPU-based Allreduce schemes.

To get further insights into the link utilization, we conduct detailed experiments using the state-of-the-art communication libraries performing Allreduce across 6 GPUs (refer to Section 5 for experimental configuration). Figure 2 depicts the result of these experiments. As we can see, none of state-of-the-art solutions neither achieve proper link utilization nor load balance communication through the available NVLinks. For instance, NVLink between GPU1 to GPU3 (G1<->G3) almost remains mostly idle, i.e., less than 1GB/s, during Allreduce operations in all state-of-the-art libraries. *In fact, the state-of-the-art collective communication library (NCCL v2.6 as of the paper is written), can only utilize 8 out of 12 NVLinks.*

These observations leads to the following broad challenge: ***Is it possible to design a link-efficient Allreduce algorithm that can maximize the utilization of available hardware communication channels to provide the best possible performance for***

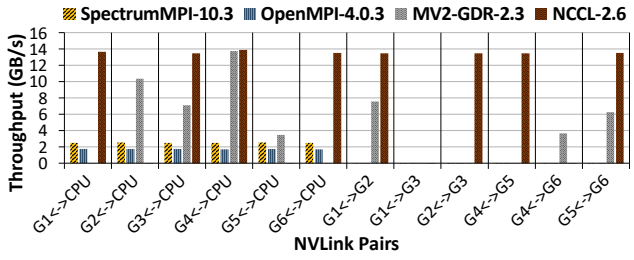


Figure 2: Under-utilization and load imbalance of all NVLinks in the state-of-the-art communication libraries on Summit system, where it has 12 NVLinks (G1-6 represent GPU1–GPU6)

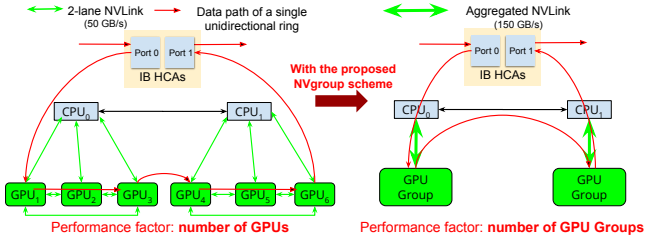


Figure 3: Expected Benefit of the proposed design on Summit system: reducing the performance factor from number of GPUs to number of GPU groups

DL training on emerging dense GPU systems with heterogeneous interconnect architectures?

1.2 Contributions

In this paper, we take up this challenge and we propose ***NVGroup*** — a link-efficient scheme through NVLink-aware cooperative reduction kernels to significantly accelerate Allreduce operations for distributed deep learning applications. As depicted in Figure 3, the proposed NVGroup design with a ring-based algorithm expects to reduce the dominant performance factor from the number of GPUs (p) to the number of GPU groups (N_{nv}), where $p > N_{nv}$. The performance evaluation and analysis demonstrate that the proposed designs can intelligently orchestrate communication to utilize all GPUs and NVLinks for performing collective operations. As a result, the proposed designs reduce 48% of time in Allreduce benchmarks up to 1,536 GPUs on the Summit system. Moreover, when performing distributed DL training using Tensorflow, the proposed design yields 19% higher throughput than the state-of-the-art solutions and it provides 172× speedup (over single GPU) using 192 GPUs, i.e., 89% scaling efficiency, on the Summit system.

To the best of our knowledge, this is the first study that deals with designs tailored for cutting-edge systems like DGX-2 and Summit-like clusters with CPU-GPU NVLinks. The key contributions of this paper are as follows.

- Provide an in-depth analysis and identify two major performance issues, namely contention and link under-utilization, for existing Allreduce solutions on modern NVLink-enabled systems. (Section 3)
- Propose a cooperative collective kernel to leverage **all** available NVLinks while the state-of-the-art scheme such as NCCL can only use 8 links (out of 12). (Section 4.2)
- Design a novel topology-aware Allreduce scheme on top of the proposed cooperative collective kernel to achieve high communication overlap among various interconnects such as NVLink, X-bus, PCIe, and InfiniBand. (Section 4.3)
- Demonstrate the proposed designs can yield up to 48% lower latency and 1.8X higher throughput for the Allreduce operation compared to state-of-the-art schemes and production libraries, including NCCL and SpectrumMPI. (Section 5)
- Provide a comprehensive performance evaluation of the proposed designs using micro-benchmarks as well as a real-world application (TensorFlow) on NVLink-enabled systems, including a DGX-2 and Summit systems. (Section 5)

Table 1: Comparison of state-of-the-art Allreduce Schemes for GPU-resident data in the literature

Bibliography Reference	Noteworthy Designs	Contention Aware	RDMA Support	GPU/NVLink-Awareness Capability and Optimization			
				Load/Store between GPUs	Utilize all NVLinks between CPUs and GPUs	NVLink Load Balance	Overlap Reduction with Communication
[34]	Kernel-based Reduction and Communication	✗	✓	✓	✗	✗	✗
[12]	Kernel-based Reduction for Recursive Doubling	✗	✓	✗	✗	✗	✗
[3]	Kernel-based Reduction for Recursive Vector Halving and Doubling	✗	✓	✗	✗	✗	✗
[24]	Topology-Aware and Event-based comm.	✓	✓	✗	✗	✗	✗
[47, 49]	Hierarchical/Multi-level Schemes	✗	✓	✗	✗	✗	✓
Baidu-Allreduce[2]	GPU-based Ring Scheme	✓	✓	✗	✗	✗	✗
Blink [50]	Broadcast-based Scheme	✓	✗	✓	✗	✗	✓
NCCL [31, 45]	GPU-based Double Binary Tree + Ring	✓	✓	✓	✗	✗	✓
BlueConnect [10, 11]	Multi-level Allreduce decomposing	✗	✓	✓	✗	✗	✗
Proposed (This paper)	**NVGroup: Link-Efficient and Cooperative**	✓	✓	✓	✓	✓	✓

2 BACKGROUND

In this section, we delve into background knowledge of cutting-edge technologies relevant to this work.

2.1 NVLink, PCIe, and InfiniBand

PCIe is the standard bus connect for high-speed components between CPU and third-party devices such as GPU and IB Host Channel Adapters (HCAs). Every generation of PCIe witnessed a doubled bandwidth, with generation-1 starting at 2GB/s and generation-4 handling as high as 16 GB/s unidirectional bandwidth. Recent architecture development shows that dense GPU systems offer promising capabilities to significantly accelerate the Deep Neural Network (DNN) training process [41]. NVIDIA has gone a step further with systems like DGX-2, offering unprecedented bandwidth of NVLink2, that provides 25GB/s unidirectional bandwidth, for inter-GPU communication within a single node. InfiniBand is an interconnect often used in the HPC community, characterized by its high bandwidth and low latency. The latest InfiniBand EDR and upcoming HDR adapters provide bidirectional bandwidth of 100Gbps and 200Gbps, respectively. With the development of interconnect technology, the next-generation HPC systems, including the #1 on Top500 *Summit* are moving toward NVLink-enabled dense GPU nodes with InfiniBand interconnects.

2.2 NVIDIA GPUDirect Technology

NVIDIA GPUDirect [30] enables direct reading and writing to CUDA host and device memory by the CPU and GPU, minimizing the latency and overhead of multiple copies to memory. This is advantageous for compute-intensive applications by providing higher throughput and lower latency through pinned buffers shared by the network and the devices. It also enables Peer-to-Peer (P2P) transfers, copying data between the memories of devices on the same PCIe bus. Aside from memory transfers between devices, it also introduced the use of remote direct memory access (RDMA) between GPUs and other PCIe devices.

2.3 CUDA-Aware Communication Libraries

MPI libraries like OpenMPI [35], SpectrumMPI [19], and MVAPICH2 [28] that can distinguish between host and GPU buffers are known as *CUDA-Aware* MPI libraries. These libraries have been designed with many optimized GPU-based point-to-point communication schemes such as staging, pipelining, CUDA Inter-Process Communication (IPC), and GPUDirect RDMA. These schemes provide the best performance across various scenarios like intra-node, intra-socket, inter-node, and several other communication paths.

NVIDIA Collective Communication Library (NCCL) is an optimized GPU-based collective communication library geared towards DL workloads [31]. NCCL's API closely resembles the MPI interface and provides communication primitives for the following: broadcast, all-gather, reduce, reduce-scatter, and all-reduce. However, the NCCL APIs are not MPI-compliant and a vast number of MPI-based applications need to be modified to use it.

3 ANALYSIS OF EXISTING SCHEMES

In this section, we cover the primary motivation of this work through an in-depth analysis of the existing Allreduce approaches for GPU-resident data. We also demonstrate the performance limitation of the current approaches.

3.1 Existing Allreduce Schemes and Associated Cost Models

The GPU-based reductions are mostly based on the legacy schemes that have been widely discussed in the literature [37, 39, 46, 51, 52], with various optimization schemes built on top of them [3, 12, 34]. Following analysis provides an insight into the performance issues of mainstream Allreduce schemes for large data sizes on the modern NVLink-enabled multi-GPU systems. Table 2 lists notations used for the models in this paper.

3.1.1 Ring-based Allreduce. In a Ring-based algorithm, processes form a logical ring structure so that each process sends and receives data to and from its logical neighbors. A naive ring-based Allreduce algorithm takes $p-1$ steps to push reduced data to all the processes.

Table 2: Notations for the Analytical Models

Name	Description
p	Number of processes
N	Number of nodes
N_G	Number of GPUs per node
N_{nv}	Number of NVLink groups in the proposed design
NV_G	Number of GPUs in a NVLink group in the proposed design
t_s	Set up time for a single send/receive operation
M	Message size (bytes)
$T_R(M)$	Time of launching a GPU kernel for reduction operation of a M-byte data
B	Bandwidth of slowest interconnect between processes
B_{NV}	Bandwidth of NVLink

An efficient ring-based Allreduce algorithm for large messages is presented in [37], and implemented in NVIDIA NCCL, AMD RCCL, and Baidu Allreduce [1, 2, 31] for GPU-resident data. It is essentially a reduce-scatter (RS), followed by an Allgather operation. Specifically, it divides the message into p smaller chunks. In the reduce-scatter phase, each process sends one chunk to its neighbor. Upon receiving a chunk from the neighbor, a process performs the reduction operation on it and proceeds with the next step using the reduced chunk. The reduce-scatter phase takes $p-1$ steps, and all processes will have a piece of the final reduced result. Similarly, the Allgather phase would take $p-1$ steps to propagate the final partial results to all the processes in the ring. The cost can be defined as follows.

$$T_{(RS\_Ring)} = (p-1) \times (t_s + \frac{M}{p \times B} + T_R(\frac{M}{p})) \quad (1)$$

$$T_{(Allgather\_Ring)} = (p-1) \times (t_s + \frac{M}{p \times B}) \quad (2)$$

$$T_{(Allred\_Ring)} = T_{(RS\_Ring)} + T_{(Allgather\_Ring)} \quad (3)$$

Although the proper chunking can amortize the communication cost, the cost of reduction operations in RS phase is inevitably sequential and dominates the overall performance. Hierarchical ring-based schemes [10, 31, 49] can be used to further reduce the dominating factor in the flat ring algorithms for multi-GPU systems. A two-dimensional ring algorithm can perform intra-node Allreduce first, followed by an inter-node Allreduce operation. As a result, the domain factor becomes $(N+N\_G)$, where $(N+N\_G) \leq p$, as shown in Equation 4.

$$\begin{aligned} T_{(Allred\_2D\_Ring)} &= T_{(Intra-node\_Ring)} + T_{(Inter-node\_Ring)} \\ &= 2 \times (N_G - 1) \times (t_s + \frac{M}{N_G \times B}) + (N_G - 1) \times T_R(\frac{M}{N_G}) \\ &+ 2 \times (N - 1) \times (t_s + \frac{M}{N \times B}) + (N - 1) \times T_R(\frac{M}{N}) \ . \end{aligned} \quad (4)$$

Note that this can be extended to higher dimensions if applicable, e.g., 3D Ring in [10, 11].

3.1.2 Tree-based Allreduce. In the ring-based algorithms, the number of steps it takes becomes the main performance bottleneck at a large scale. To further reduce the communication cost while maintaining the full bandwidth at scale, the state-of-the-art adopts a tree-based design for GPU-based Allreduce [45]. The basic idea is to adopt a double binary tree algorithm (DTree) proposed in [40] for reduction and broadcast operations, which can be combined into an Allreduce operation. By building two binary trees with each tree responsible for Allreduce operations of half the data in different directions, we can fully exploit the duplex interconnects. Ideally, the dominant factor is expected to be reduced to logarithmic with a proper pipeline, as shown in Equation 5.

$$\begin{aligned} T_{(Allred\_DTree)} \simeq 2 \times (\log_2 p) \times (t_s + \frac{M}{2 \times p \times B}) \\ + (\log_2 p) \times T_R(\frac{M}{2 \times p}) \ . \end{aligned} \quad (5)$$

3.1.3 Rabenseifner's algorithm for Allreduce. Rabenseifner's algorithm, also known as Recursive vector halving and doubling (RVHD), is a tree-like algorithm proposed as an alternative to the ring-based described above for large messages at scale. RVHD can be applied to Allreduce, which is again essentially reduce-scatter followed by Allgather [39, 46, 51]. In the reduce-scatter phase, each process first exchanges the data with a process that is a distant $\frac{p}{2}$ away. Here, each process sends half of the message and receives another half of the message, and then performs the reduction operation on the received message. In the following steps, the distance and the message size to be exchanged are halved in each step. The reduce-scatter phase takes $\log_2 p$ steps. In contrast, in the Allgather phase, the distance and the message size to be exchanged are doubled in each step. Based on [46], the cost of RVHD Allreduce can be represented as follows:

$$\begin{aligned} T_{(RS\_RVHD)} &= \log_2 p \times t_s + \frac{p}{p-1} \times \frac{M}{B} \\ &+ T_R(\frac{p}{p-1} \times M) \\ T_{(Allgather\_RVHD)} &= \log_2 p \times t_s + \frac{p}{p-1} \times \frac{M}{B} \end{aligned}$$

$$T_{(Allred\_RVHD)} = T_{(RS\_RVHD)} + T_{(Allgather\_RVHD)} \quad (6)$$

Although RVHD is proven to achieve lower bound of Allreduce when p is power of two [39], it may not be the case for modern GPU systems such as Summit, which has 6 GPUs per node.

3.2 Performance Issues of Existing Allreduce Algorithms on NVLink-enabled Systems

This section includes an analysis and discussion of the problems with the existing Allreduce schemes on the NVLink-enabled multi-GPU systems. Figure 4 illustrates an intermediate step of the tree-based Allreduce schemes being applied to multi-GPU systems. Contention can easily occur on the slower interconnects such as IB networks or X-bus while every GPU is exchanging data with another process across the nodes or the sockets. This has been observed in host-based Allreduce on symmetric multiprocessing (SMP) scenarios as discussed in [37].

To address the contention problem, the ring-based Allreduce schemes are proposed. Figure 5 shows an intermediate step of the ring-based Allreduce scheme on a multi-GPU node. Although this scheme is contention-free, it only utilizes all the links in one direction whereas all the modern interconnects support full-duplex links. Current ring-based Allreduce designs mostly use the uni-directional ring to avoid contention on traditional PCIe-based systems. With the introduction of NVLink, researchers have started proposing multiple rings to saturate the bandwidth of interconnects in both directions. However, a maximum of two rings are used to avoid contention in the slower interconnects. If there are more than two GPUs connected to the CPU similar to the Summit system [33], some NVLinks between the CPU and the GPU remain idle and are never used, as shown in Figure 5. This leads to lower link utilization and load imbalance among NVLinks as depicted in Figure 2.

In short, the state-of-the-art Allreduce schemes are suffering from either contention or link under-utilization on high-performance NVLink-enabled multi-GPU systems. There is a dearth of topology-aware Allreduce schemes that are contention-free while maximizing the use of fast interconnects like NVLinks.

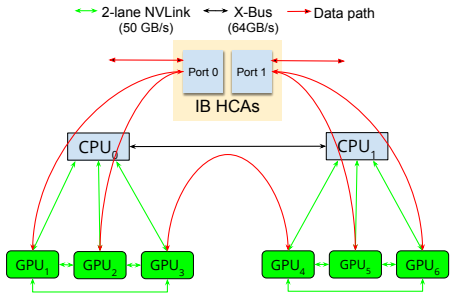


Figure 4: Flat Tree-based Allreduce algorithms. At any given step, each GPU exchanges a vector with another GPU based on a logic tree. Contention occurs on the slowest interconnects like InfiniBand networks.

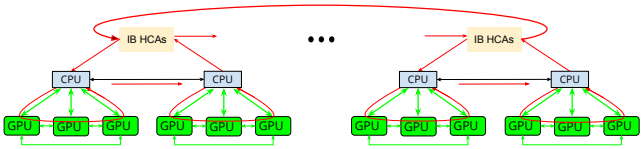


Figure 5: Contention-free Ring-based Allreduce but low link-utilization. At any given step, each GPU sends and receives chunks to and from its neighbors. Links are either not fully utilized on both directions or remain idle.

4 PROPOSED LINK-EFFICIENT DESIGNS ON NVLINK-ENABLED GPU SYSTEMS

Based on the analysis presented in Section 3, three major challenges must be addressed to achieve the best possible performance on emerging NVLink-enabled dense GPU systems: 1) utilizing all NVLinks between CPU and GPU and between GPUs, 2) avoiding contention on the slower interconnects, and 3) hiding the cost of "reduction" as much as possible. In this section, we present our proposed designs to address these challenges.

There are three main components: ❶ NVLink discovery and grouping, ❷ Cooperative Collective communication within a NVLink group, and ❸ Communication across NVLink Groups.

4.1 NVLink Discovery and Grouping

The modern operating systems and CUDA drivers provide methods such as NVIDIA Management Library (NVML) [29] to dynamically query the information of the device (e.g., CPU, GPU or HCA) and interconnect (e.g., NVLink or PCIe bus). NVML, combined with tools such as the Portable Hardware Locality library (hwloc) [8], help to determine the locality information for the CPU, GPU, and HCA. These tools are useful in easily generating a topology graph. An example of a topology graph we built with these tools is shown in Figure 3. In principle, we *group* the CPU cores and GPUs that are fully connected by NVLinks such that within an NVLink group (NVGroup), any GPU can access other GPUs' memory and system memory via a dedicated NVLink without contention from communication operations originating from other GPUs. This grouping provides an abstraction to the CPU processes that it connects with a single powerful GPU via fast NVLink. Next, an Allreduce algorithm can be built based on these groups instead of individual GPUs. Note that this is different from the typical NUMA-aware or hierarchical approaches [10, 24, 47] because NVLinks can be used to connect GPUs across physical sockets. For instance, in TSUBAME3 [48] and DGX-2, we will have one single NVLink group per node even though two NUMA nodes exist because NVLink/NVSwitch fully connects all GPUs.

4.2 Persistent GPU Kernels for Cooperative Allreduce within an NVLink Group

To create an abstraction of a single powerful GPU for CPU processes, GPUs must work cooperatively. In this work, we adopt a persistent GPU kernel approach [13, 16, 17, 31] to avoid the overhead of multiple CUDA kernel launches. The proposed collective persistent kernel (COLL-PK) acts as a worker for CPU processes. There are three primary design considerations for such a persistent kernel: 1) Low-overhead signaling mechanism, 2) Efficient collective operations within NVGroup, and 3) High kernel occupancy and NVLink utilization.

4.2.1 Signaling Mechanism. Since the CPU process is responsible for orchestrating data transfer via the IB network, a signaling mechanism is required for the CPU to notify the GPU to work on new requests and also to query the completion of the work requests. The proposed design uses a pre-allocated circular buffer in the system memory, shared by all CPU processes and GPUs in the same group, as a work queue to store the work requests. In general, we create one work queue for each Cooperative Thread Array (CTA) to avoid additional synchronization between CTAs. If an advanced CUDA cooperative group feature is used [25], only one work queue is sufficient. Each work request contains four fields: request ID, collective type, data offset, and data size. Furthermore, CPU maintains a *request counter* to indicate the request number of the outstanding work request. In the GPU kernel, each CTA maintains a *response counter* to indicate the completion of the request number, similar to the cumulative acknowledgment. To minimize synchronization overhead between CPU and GPU, only one thread in each CTA keeps polling the *request counter* and reads the request item to the shared memory (within the CTA) if a new work request is ready as show in Algorithm 1 (lines 10-12).

4.2.2 Collective Operations within NVGroup. To support Allreduce operations, the proposed COLL-PK provides varieties of basic collective operations among GPUs: reduction and broadcast. To maximize the GPU computing power and minimize synchronization between GPUs, each GPU within a group is responsible for performing collective operations on different portions of the data. Algorithm 1 highlights the main components of the COLL-PK running on the GPU. To achieve this, the CPU process assigns a work request to each GPU and its CTAs with a different offset in the work request.

Within one GPU, we further split the data and assign them to different CTAs to get high occupancy and work balance as shown in Figure 6. Most of the state-of-the-art communication libraries that implement reduction and broadcast in a so-called *Copy-Compute-Forward* fashion, which performs a copy of the whole buffer followed by the reduction operation and finally forwards it to other GPUs or CPU. However, explicit data transfer is expensive. The proposed COLL-PK performs the reduction in a *Load-Compute-Store* via NVLink. That is, each GPU thread loads an element, e.g., words

Algorithm 1 The proposed cooperative persistent reduction kernel

```
   Definition of Variables
1:    nvgroup : a set of GPUs in the NVGroup.
2:    buf_gpu : the buffers of GPU memory within the NVGroup.
3:    buf_shm : the buffers of shared system memory within the NVGroup.
4:    req : request array for CTAs to perform communication.
5:    groupID : group ID of current GPU in the NVGroup.
6:    bid : block/CTA ID.
7:    tid : thread ID.
   KERNEL RUNNING ON EACH CTA:
8:    has_more_req ← 1
9:    while has_more_req == 1 do
10:       while tid==0 AND req[bid].request_cnt <= req[bid].response_cnt do
11:          /* Wait for new request(s) from CPU process */
12:       end while
13:       _block_sync_();
14:       switch req[bid].action do
15:          case LOAD-REDUCE-STORE-TO-LOCAL:
16:             // LOAD and REDUCE on buf_gpu within nvgroup
17:             // STORE the result into buf_gpu[groupID]
18:             ++req[bid].response_cnt;
19:             break;
20:          case LOAD-REDUCE-STORE-TO-CPU:
21:             // LOAD and REDUCE within nvgroup
22:             // STORE the result into buf_shm[groupID]
23:             ++req[bid].response_cnt;
24:             break;
25:          case LOAD-REDUCE-STORE-TO-GROUP:
26:             // LOAD and REDUCE on buf_gpu within nvgroup
27:             // STORE the result into buf_gpu within nvgroup
28:             ++req[bid].response_cnt;
29:             break;
30:          case GROUP-BCAST:
31:             // LOAD data from buf_shm[groupID]
32:             // STORE/broadcast the data to buf_gpu
33:             ++req[bid].response_cnt;
34:             break;
35:          case TERMINATE:
36:             has_more_req ← 0
37:             break;
38:    end while
```

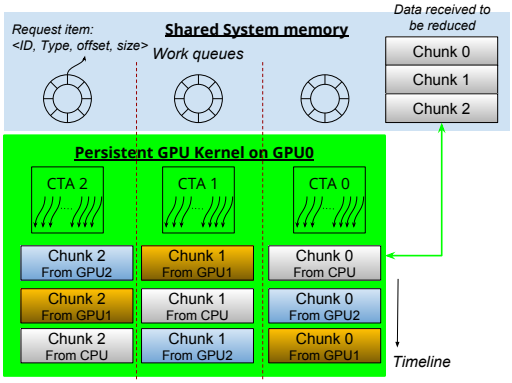


Figure 6: Example of Persistent GPU kernel Performing Cooperative Allreduce Operations among three GPUs

of size equal 16 bytes to maximize the memory access throughput, from CPU and remote GPU via NVLink, and performs the reduction on the element, followed by a direct store instruction to move the element to CPU or remote GPUs. In this way, the reduction and broadcast operation can be highly overlapped, maximizing the throughput. As an example shown in Figure 6, each CTA is working on a different data chunk. Each CTA can concurrently LOAD data from different GPUs in the NVGroup via different NVLinks, so that all the collective operations are overlapped.

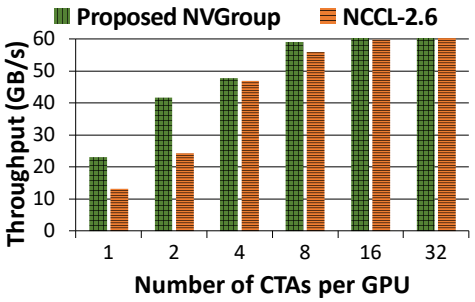


Figure 7: Peak Allreduce throughput of the proposed persistent kernel with various number of CTAs per GPU on a POWER9 system with three GPUs in one group

4.2.3 GPU Occupancy and NVLink Utilization. Achieved occupancy, "the ratio of active warps/threads on an streaming multiprocessors (SM) to the maximum number of active warps/threads supported by the SM", is a key factor for a high-performance GPU kernel. The state-of-the-art communication library like NCCL builds the ring structure inside the GPU kernel and uses CPU threads to act as a worker to perform communication across nodes. However, such a design consumes a vast amount of valuable GPU registers in the SM and prevents us from utilizing more threads in parallel. For example, the GPU reduction kernel of state-of-the-art NCCL is limited by 256 threads per CTA. The proposed COLL-PK stores the algorithm-related information on the system memory to enable all the GPU threads, e.g., 1,024 threads per CTA in NVIDIA Volta GPU, available for performing reduction and data movement operations. Next, we need to ensure COLL-PK has enough working threads to saturate the NVLink while preserving resources for the applications. Figure 7 shows the peak throughput of 128-MByte Allreduce operation with various numbers of CTAs per GPU on a POWER9 system with two NVGroups and three GPUs per group. The proposed COLL-PK can reach 60GB/s by using only 8 CTAs and 1024 threads per CTA. Note that the latest NVIDIA Volta GPU is capable of scheduling up to 160 CTAs concurrently. There are two significant benefits by having fewer CTAs. First, fewer CTAs brings lower kernel launch and synchronization overhead due to the fewer CTAs to be scheduled and synchronized. Second, it leaves more GPU resources, i.e., SM, for applications to perform other useful computations that can be overlapped with All-reduce operations. As shown in Figure 7, the throughput of the proposed COLL-PK kernel using 8 CTAs is sufficient to fully utilize all interconnects such as NVLink, PCIe, X-bus, and InfiniBand. Therefore, we use this setting for the rest of the paper.

4.3 Group-wise Allreduce Communication

If there is more than one group, we can perform allreduce algorithms like ring-based or tree-based across groups. Here, we take a ring-based algorithm to illustrate how to design the algorithm on top of NVGroup. Within the NVGroup, all available NVLinks are used to perform intra-group Allreduce cooperatively as described in Section 4.2. The CPU orchestrates inter-group communication, and multiple duplex HCAs can be used for data transfer without contention to maximize the bandwidth. We elaborate on the various phases of the algorithm in the following sections.

4.3.1 Reduce-Scatter Phase. In the first step, each group computes a "reduced" chunk within the group as described in Section 4.2 and stores data directly on the system memory shared by the processes within the group. Note that each GPU pushes data to system memory via their dedicated NVLink between GPU and CPU. The cost of this step (Step 3 in Algorithm 2) can be formulated as follows:

$$2 \times \frac{M}{N_{nv} \times NV_G \times B_{NV}} + T_R(\frac{M}{N_{nv} \times NV_G}) \quad (7)$$

Next, the group leader starts "filling the pipe" of the ring by pushing the "reduced" chunk, which is $\frac{M}{N_{nv}}$ bytes, to the next neighbor in the ring. Here, the communication cost can be represented as $\frac{M}{N_{nv} \times B}$, which is bounded by the slowest interconnects such as IB networks. Meanwhile, GPUs within the group remain busy preparing the reduced chunk in the GPU memory for the communication in the next step. Note that the cost of preparing the reduced chunk shown below is completely "hidden" because the communication cost is significantly higher than Allreduce operation across GPUs via NVLink in the modern architectures. Also, each GPU is only responsible for a portion of the chunk, which is $\frac{M}{N_{nv} \times NV_G}$ bytes. Upon receiving a chunk from the neighbor, the group leader notifies GPUs in the group by updating counters in the shared memory to signal the persistent kernel as described in Section 4.2.1. Then, all GPU threads perform LOAD-REDUCE-STORE via their individual NVLinks without contention. Since the communication of the next step is dependent on the same data chunk, this reduction operation cannot be hidden. After $(N_{nv} - 1)$ steps, each group leader holds a different chunk with final reduced results in the system memory. In summary, the cost of the Reduce-Scatter phase (Step 4 in Algorithm 2) can be formulated as follows:

$$T_{(RS\_NVGroup-Ring)} = (N_{nv} - 1) \times (t_s + \frac{M}{N_{nv} \times B} + T_R(\frac{M}{N_{nv} \times NV_G})) \quad (8)$$

4.3.2 Allgather Phase. The allgather phase is similar to the reduce-scatter phase, but without any computation. Upon receiving a new chunk, the group leader notifies GPUs in the group to pull the data via NVLinks. Then, the group leader pushes the data to the next neighbor over networks without synchronizing with the GPUs. The cost of the allgather phase (Step 5 in Algorithm 2) can be expressed as follows:

$$T_{(Allgather\_NVGroup-Ring)} = (N_{nv} - 1) \times (t_s + \frac{M}{N_{nv} \times B}) \quad (9)$$

Finally, the cost model of the proposed NVGroup with a ring algorithm can be represented as follows. Note that it looks similar to Equation 3, but the number of steps is reduced from the number of processes p to the number of NVLink groups N_{nv}.

$$T_{(Allred\_NVGroup-Ring)} = Eq\ 7 + Eq\ 8 + Eq\ 9 \quad (10)$$

In this paper, we use the ring-based Allreduce algorithm as the example to demonstrate the efficacy and benefit of the proposed mechanisms of NVLink grouping and the persistent GPU kernel for modern multi-GPU systems. Please note that the proposed design can apply to other algorithms and collective operations as well.

Algorithm 2 Overview of NVGroup-Ring Allreduce

```
   Definition of Variables
1:   buf_gpu : the device buffers within the NVGroup.
2:   buf_shm : the system memory shared within the NVGroup.
3:   req : request array for CTAs to perform communication.
4:   gID : group ID of current GPU in the NVGroup.
   ROUTINE RUNNING ON EACH MPI PROCESS:
5:   Step 1: NVLink Discovery and Grouping (Section 4.1).
6:   chunkSize ← M / N_nv;
7:   Step 2: Launches the persistent reduction kernel (Algorithm 1) on GPU.
8:   Step 3: Signal kernel start performing reduction for the chunks about to send
9:     for chunkID ← 0 to N_nv do
10:      req[gID].action ← LOAD-REDUCE-STORE-TO-LOCAL;
11:      req[gID].chunkID ← chunkID;
12:      req[gID].chunkSize ← chunkSize;
13:      ++req[gID].request_cnt;
14:    end for
15:  Step 4: /* Reduce-scatter phase for all chunks */
16:    if isGroupLeader then
17:      /* leader process posts receives for all chunks */
18:      for chunkID ← 0 to N_nv do
19:        MPI_Irecv(buf_shm[chunkID], chunkSize, fromLeftGroup);
20:      end for
21:    end if
22:    while N_nv chunks are not yet all reduced do
23:      if chunk_i is received then
24:        /* Signal GPU to perform local reduce for the chunk */
25:        req[gID].action ← LOAD-REDUCE-STORE-TO-CPU;
26:        req[gID].chunkID ← i;
27:        ++req[gID].request_cnt;
28:      end if
29:      if Any chunk_j is received and reduced && isGroupLeader then
30:        MPI_Isend(buf_shm[j], chunkSize, toRightGroup);
31:      end if
32:    end while
33:  Step 5: /* Allgather phase for all chunks */
34:    if isGroupLeader then
35:      /* leader process posts receives for all chunks */
36:      for chunkID ← 0 to N_nv do
37:        MPI_Irecv(buf_shm[chunkID], chunkSize, fromLeftGroup);
38:      end for
39:    end if
40:    while N_nv chunks are not yet all reduced do
41:      if chunk_i is received then
42:        /* Signal GPU to broadcast the chunk within the group */
43:        req[gID].action ← GROUP-BCAST;
44:        req[gID].chunkID ← i;
45:        ++req[gID].request_cnt;
46:        if isGroupLeader then
47:          MPI_Isend(buf_shm[j], chunkSize, toRightGroup);
48:        end if
49:      end if
50:    end while
```

4.4 Optimizations

4.4.1 Fine-grained Pipeline for High Communication Overlap. The proposed topology-aware *Allreduce* scheme essentially consists of three stages during the communication, 1) NVLink for LOAD-COMPUTE-STORE between GPU to GPU, 2) NVLink for LOAD-COMPUTE-STORE between CPU and GPU, 3) InfiniBand for pure data movement between GPU nodes. During the time this paper was written, the throughput of the IB network is significantly slower than NVLink, e.g., about 2-3X. To achieve optimal performance, one needs to keep the slowest IB network busy. In general, the reduce-scatter and allgather phases would divide the data into N_{nv} in the proposed NVGroup scheme (p chunks in existing algorithms). However, for a very large message size, the chunk size could be large to keep NVLinks busy and delay the data transfer through IB (i.e., IB remains idle). To avoid such a scenario, we add one more

layer of 'chunking' to start 'filling the pipe' earlier and improve the overall performance.

4.4.2 CPU-driven Allreduce for Small Messages. Since the GPU-based solution is throughput-optimized, it falls short when the amount of data is small. When the message size is too small to benefit from the ring-based algorithm and persistent kernels, e.g., GPU kernel launch overhead is greater than the GPU computation time [12], we propose using a CPU-driven Allreduce scheme: 1) use CPU to perform reduction operations on the shared system memory when the data chunk is too small, e.g., <= 16KB, for each NVGroup to saturate NVLink, and 2) CPU-based recursive doubling algorithm for large-scale and small-message Allreduce where the CPU copies the data between GPU to system memory using low-latency copy schemes [43]. As a result, we can avoid the expensive kernel launch overhead for reducing the latency of short-message reduction.

5 PERFORMANCE EVALUATION

In this section, we detail the performance evaluation and include an analysis of the proposed designs.

5.1 Experimental Platforms and Setup

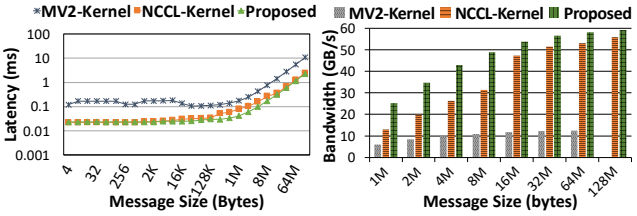
We conducted experiments on the following cutting-edge NVLink-enabled GPU platforms:

- *NVLink-based IBM OpenPOWER9 systems*: Each node has two POWER9 CPUs. Every socket has 22 IBM Power9 cores with four hardware threads per core. The CPU sockets are connected using X-bus interconnect. We evaluated the proposed design on two types of POWER9 systems: 1) *Summit* system [33]: three NVIDIA V100 GPUs per socket and 3-lane NVLink is used to connect CPU and GPU and between GPUs on the same socket, 2) *Lassen* system [23]: there are two NVIDIA V100 GPUs per socket and 2-lane NVLink.
- *NVIDIA DGX-2 AI system*: The DGX-2 system is very different compared to the POWER systems. It has an advanced topology with up to 6 NVLink(s) between 16 NVIDIA Telsa V100 GPUs connected using the NVSwitch.

We evaluated the proposed and existing Allreduce schemes as follows: 1) NVGroup-Ring (*Proposed*), which is implemented in MPI_Allreduce on top of the MVAPICH2-GDR library, 2) a GPU-based Rabenseifner's algorithm implemented in MVAPICH2-GDR v2.3 [3] (*MV2-GDR-2.3*), 3) Ring- and tree-based implementations in NCCL v2.6 [45] (*NCCL-Ring* and *NCCL-Default*) where double binary tree algorithm is used by default and for inter-node communication only. Note that previous academic work are only comparing to *NCCL-Ring* [10, 11, 49]. We also compared the proposed design with production libraries such as SpectrumMPI v10.3.1 that is only available on POWER systems (*SpectrumMPI*), and OpenMPI v4.0.3 with UCX v1.8 (*OpenMPI*). Note that SpectrumMPI and OpenMPI are production communication libraries that dynamically select the best possible communication schemes for given system scales and message sizes; we evaluated them when it is available.

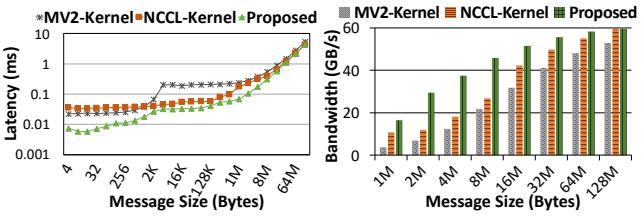
5.2 Benchmark-level Evaluation

The evaluation and analysis are using micro-benchmark, which is a modified benchmark suite from NCCL-test [32] to support



(a) Latency (b) Peak Bandwidth

Figure 8: Performance Comparison of Allreduce kernel over NVLink for Proposed vs. state-of-the-arts on Summit system: Three GPUs and CPU are fully connected



(a) Latency (b) Peak Bandwidth

Figure 9: Performance Comparison of Allreduce kernel over NVLink for Proposed vs. state-of-the-arts on DGX-2 system: 16 GPUs are fully connected

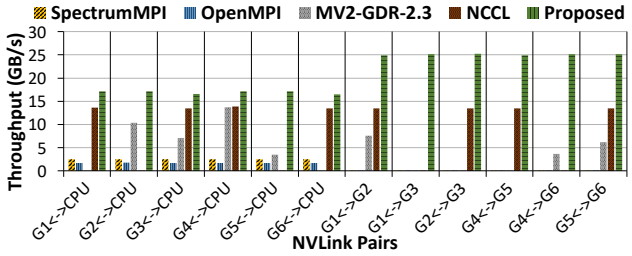
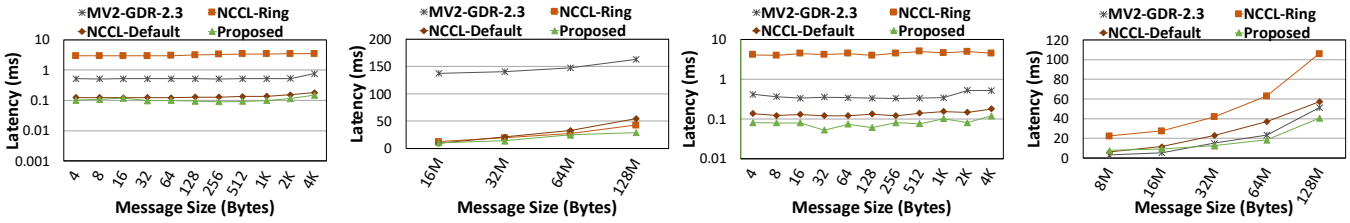


Figure 10: NVLink Utilization: Allreduce performance for NVGroup-Ring (Proposed) vs. state-of-the-art communication libraries on Summit system (G1 represents GPU1 and the same for others)

collectives using both NCCL and MPI primitives, i.e., ncclAllreduce and MPI_Allreduce. The primary performance metrics are latency and bandwidth of a single Allreduce operation. All the numbers reported in this section are the averages of three runs and 100 iterations per run including ten warm-up iterations.

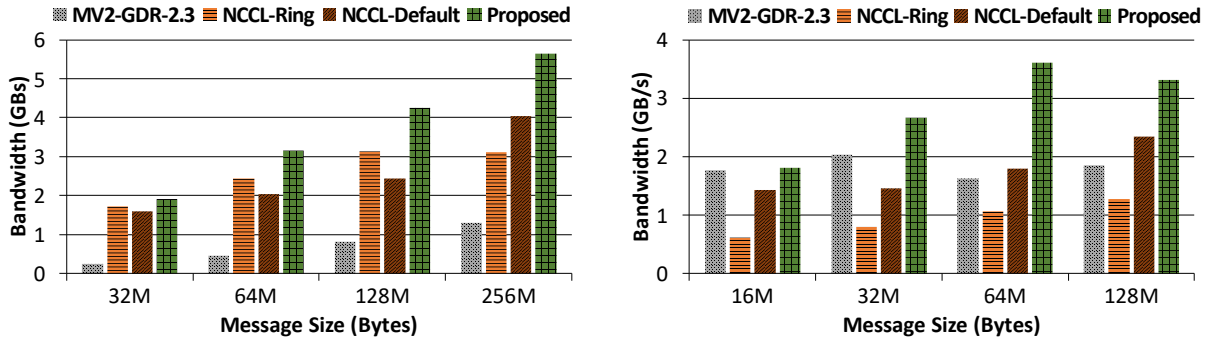
5.2.1 Cooperative Reduction Kernel over NVLink. To understand the efficiency of the proposed cooperative reduction kernel over NVLink, we conducted the experiments on the GPUs that are fully connected by NVLink. Without interference from slower interconnects, we can get insight into the NVLink utilization of different Allreduce schemes. Here, we compared the proposed kernel to existing GPU-enabled reduction schemes such as NCCL (*NCCL-Kernel*) [31] and MVAPICH2-GDR (*MV2-Kernel*) [12] that have been widely used in the literature [3, 11, 49, 50].

Figures 8(a) and 8(b) show the performance comparison of Allreduce operation on three GPUs on the Summit system. In this configuration, all GPUs and the CPU are fully connected by the dedicated



(a) 1536 GPUs (6 GPUs/Node) on Summit (b) 512 GPUs (4 GPUs/Node) on Lassen

Figure 11: Latency Comparison: Allreduce performance for the proposed vs. state-of-the-art schemes on NVLink-enabled POWER systems



(a) 1536 GPUs (6 GPUs Per Node) on Summit (b) 512 GPUs (4 GPUs/Node) on Lassen

Figure 12: Bandwidth Comparison: Allreduce micro-benchmark performance for NVGroup-Ring (Proposed) vs. state-of-the-art schemes on NVLink-enabled POWER systems

NVLink. In Figure 8, we see that the proposed design outperforms existing communication libraries in all message sizes as a result of the efficient Allreduce kernels with high NVLink utilization. For Allreduce operations of 128MB data, the proposed design has 1.9X shorter latency (i.e., reduces 48%) and almost 2X higher bandwidth compared to the NCCL-Kernel.

Figure 9 shows the micro-benchmark performance of Allreduce operations for 16 GPUs on the DGX-2 system. The proposed design can offer the best performance across the entire message range compared to other MPI libraries, where the proposed kernel design (Proposed) is up to 6X faster than the MV2-Kernel. Finally, the proposed design outperforms NCCL-Kernel up to 2.5X for message sizes smaller than 16-MByte because of the CPU-driven optimizations and the efficient GPU kernel. For large messages shown in Figure 9(a), the proposed design can saturate the NVLink significantly faster than existing schemes. For messages larger than 64 MB, both NCCL and the proposed designs are saturating all the NVLinks; thus yielding comparable performance.

5.2.2 Link-Utilization and Load Balance. To address the key problem of link under-utilization and load imbalance, as illustrated in Figure 2, Figure 10 presents new results derived from the proposed design. The benefits observed for the proposed NVGroup-Ring are corroborated by the fact that the proposed designs are aggressively utilizing all the NVLinks, whereas other libraries significantly fall behind. By profiling the NVLink counters with NVIDIA CUDA Profiling Tools Interface (CUPTI) according to [21], we observe that NVGroup-Ring (Proposed) can achieve 16GB/s and 25GB/s on GPU-CPU and GPU-GPU NVLinks, respectively. Note that the throughput is limited by the slower X-Bus interconnects.

5.2.3 Large-Scale Evaluation. The parallel applications usually perform collectives across large number of nodes to gain the best possible acceleration. For the traditional HPC applications, Allreduce is typically used for small messages, e.g., from 4 to 32 bytes. On the other hand, a large message Allreduce is highly demanded by data-parallelism-based deep learning applications to exchange and update gradients [3]. To understand the performance improvement of the proposed designs for both HPC and DL applications at scale, we report the performance comparison for small and extra-large message ranges on POWER systems up to 1,536 GPUs as shown in Figure 11. For Allreduce operations with small data as shown in figures 11(a) and 11(b), the proposed design provides lower latency as it uses the CPU-driven Allreduce as described in Section 4.4. Figures 11(a) and 11(b) report the latency comparison at scale and the proposed designs can achieve up to 3.3X, 1.8X, and 2.1X faster Allreduce compared to MV2-GDR, NCCL-Ring, and NCCL-Default, respectively. Note that BlueConnect [11], known as IBM PowerAI solution, claims to outperform NCCL-ring by 1.6X in a similar configuration, whereas the proposed design yields 20% more performance, achieving 1.8X improvement over NCCL-ring. Next, figures 12(a) and 12(b) demonstrate the bandwidth of Allreduce for large message sizes. Similar to the latency evaluation, we also observe significant improvement for bandwidth. Note that MVAPICH2-GDR uses the tree-based Allreduce algorithm [3] to perform better for smaller messages at scale. However, as shown in Figure 13, it performs worse due to a larger number of GPUs existing per node on Summit. This leads to the severe contention discussed in Section 3.2.

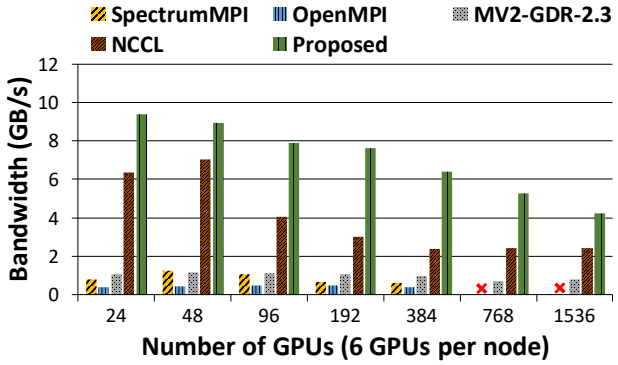


Figure 13: Scalability Evaluation: Allreduce Bandwidth of 128MBytes data for Proposed vs. state-of-the-art communication libraries on Summit system. (A red "X" indicates cases where the libraries failed to complete the experiments.)

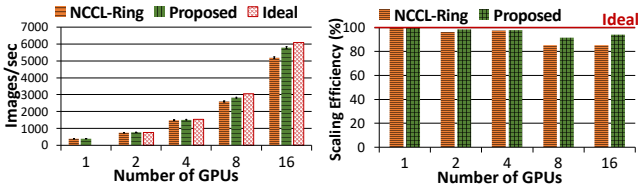
Finally, we present the scalability evaluation by fixing the message size in 128 MBytes, which is a typical use case for DL applications [3]. Here, we also compared the proposed design with production communication libraries, including SpectrumMPI and OpenMPI. As can be seen in Figure 13, the proposed NVGroup-Ring design provides higher *algorithm* bandwidth in all system sizes. The bandwidth of the all designs decreases as the GPU count increases due to the factor that more steps are required to complete the communication. The production libraries such as SpectrumMPI and NCCL typically apply different algorithms, e.g., tree- or ring-based, to enhance scalability. Yet, the proposed NV-Group design with the ring can still outperform the these production libraries in all configurations we have experimented with.

5.3 Application-level Evaluation

We now present the application level performance for our proposed design using popular distributed DL frameworks including TensorFlow (v1.14.0), PyTorch (v1.5.0), and Horovod (v0.19.1). Horovod [42] is one of the most efficient and the most straightforward design to enable distributed training for DL frameworks such as TensorFlow [3], PyTorch [36] and MXNet. It heavily relies on Allreduce communication according to [5]. The feature of tensor fusion in Horovod allows batching tiny Allreduce operations into one Allreduce to better utilize high-speed interconnects [42]. In this section, we show the performance of end-to-end DNN training with a synthetic image dataset to prevent the effect from disk I/O access. We conducted training using the ResNet-50 [18] model with a fixed batch size, 64 per GPU, using Horovod with NCCL Allreduce and Horovod with the proposed Allreduce kernel integrated into MPI Allreduce. [1]

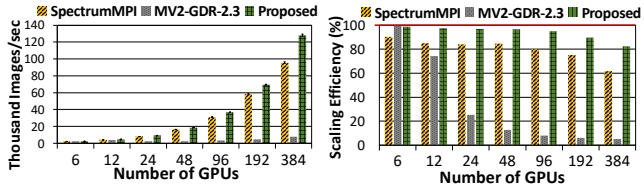
Figures 14 shows the training performance on an NVIDIA DGX-2 machine with 16 NVIDIA Volta GPUs fully connected by NVLink and NVSwitch. The images/second in Figure 14(a) provides absolute training performance numbers within 1% variation over five runs where the proposed NVGroup-Ring offers 5,788 images/sec. This is

[1]Due to the instability of TensorFlow and PyTorch frameworks on OpenPOWER systems, we could not obtain full set of performance numbers at the time this paper is written. Any further updated graphs are planned to be reported on the project website (http://hidl.cse.ohio-state.edu/) when available.



(a) Throughput (b) Scaling Efficiency

Figure 14: ResNet-50 Training using TensorFlow on DGX-2 System: NCCL2 vs. Proposed



(a) Throughput (b) Scaling Efficiency

Figure 15: ResNet-50 Training using TensorFlow on Summit system (6 GPUs/node)[1]

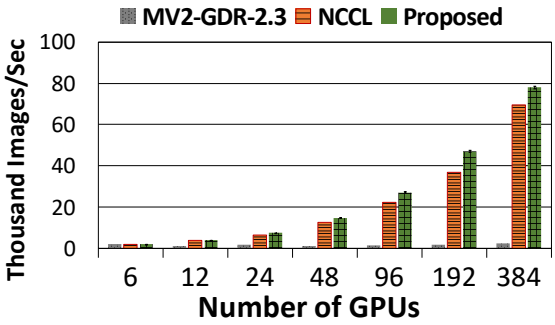


Figure 16: ResNet-50 Training using PyTorch on Summit system (6 GPUs/node)[1]

8% higher than NCCL2's 5,193 images/sec for ResNet-50 training on 16 Volta GPUs. As shown in Figure 14(b), although both designs perform similarly for 2 and 4 GPUs, the proposed design achieves 93.9% scaling efficiency (15X speedup) for 16 Volta GPUs on the DGX-2 system, which is a significant improvement over NCCL's 85.2% efficiency (13.6X speedup). The performance gains shown for the proposed designs are further justified by the micro-benchmark performance improvements reported earlier in Section 5.2.1 since the message sizes used for Allreduce operations are ranging from 256-Byte to 256-MByte in the DL training phase.

Similarly, we can observe near 90% scaling efficiency (172X speedup) on the Summit system, as shown in Figure 15(b). Figure 15(a) the proposed design is achieving 69,537.94 images/sec over 192 NVIDIA Volta GPUs. In similar experiments with PyTorch, as shown in Figure 16, we can see that the proposed design outperforms NCCL by 1.28X in training ResNet-50 over 192 GPUs. However, we note that only improving Allreduce communication is not enough to achieve ideal scaling for the PyTorch framework due to the different communication patterns being used. Further

investigation and optimization are required for different DL frameworks, and it is a part of our future work. Note that the MV2-GDR scheme has poor scalability in both cases due to severe contention, as discussed previously.

6 RELATED WORK AND DISCUSSION

Different research studies with a focus on improving large message communication exist in the literature. Chu et al. [12] proposed new MPI reduction collective designs that benefit from the CUDA kernels and GPUDirect RDMA features for reduction and communication, respectively. However, the designs are *COPY-COMPUTE-COPY* approach and do not take link utilization into account. Oden et al. [34] suggested using a Global GPU Address Space to bypass CPU involvement, enabling communication and computation to be done on the GPU for Reduce and Allreduce operations. However, it has significant driver overhead on address translation, and it requires significant modifications for MPI applications to leverage the design. Luo et al. [24] offered a new communication framework on top of OpenMPI that relaxes synchronization dependencies by using events and callbacks and utilize the CUDA kernel to speed up reduction operations. Blink is a design protocol for collective communication for obtaining better NVLink utilization on DGX-like systems[50]. They propose using two-stage protocol: internal broadcast and cross-group forwarding to fully utilize NVLink within a DGX machine. However, it does not handle communication across multiple GPU nodes. In work done by Mojumder et al. [27], the training of various DNNs is evaluated and compared based on two different communication methods: peer-to-peer data transfer and using NCCL based communication. They provide an outlook on the bottlenecks and factors that are limited by the multi-GPU system architecture.

In [22], gradient reduction strategies are applied into Horovod framework to optimize the overlap between computation and Allreduce communication. The authors demonstrate a scalable training, i.e., 93% scaling efficiency, using modified DNN segmentation model on Summit system. This work and the proposed designs are orthogonal and complimentary. Thus, they can be applied at the same time, which is part of our future work. Cho et al. [10] proposed a communication library referred to as BlueConnect for DL training that is optimized for GPU architectures. The central idea behind the efficiency proposed by this library involves decomposing Allreduce operations into multi-level reduce-scatter and All-gather operations that are parallelizable at each level. However, the operations between different levels cannot be performed concurrently due to the data dependency. The proposed NVGroup can concurrently make progress of communication across groups and computation within a group to yield performance improvement compared to NCCL's ring- and tree-based design, as shown in Section 5.

7 CONCLUDING REMARKS

As emerging GPU systems become denser with faster interconnects like NVLink, it is challenging to achieve link-efficient and contention-aware communication and to translate the performance to the applications. In this paper, we took up the challenge of designing efficient communication algorithms of the critical Allreduce collective operation for distributed deep learning training on NVLink-enabled dense GPU systems. We analyzed the deficiencies of existing algorithms for Allreduce and presented a comprehensive theoretical model. Based on this model, we proposed link-efficient, cooperative, and topology-aware algorithms for Allreduce. Using CUPTI, we analyzed and demonstrated that the proposed "NVGroup" approach can utilize the available bandwidth of NVLinks much more efficiently than the state-of-the-art communication libraries like NVIDIA's NCCL and CUDA-Aware MPI libraries. By maximizing utilization of all available NVLinks we present 1.8X faster Allreduce compared to the state-of-the-art NCCL at micro-benchmark-level on Summit system with 1,536 GPUs. Furthermore, we observe 93.9% scaling efficiency (15X speedup) and up to 8% higher throughput for training deep learning models using TensorFlow when comparing the proposed design to the existing scheme on a DGX-2 system. A similar trend is also observed on the Summit system, displaying 89.7% scaling efficiency (172X speedup) over 192 GPUs. With the PyTorch framework, the proposed NVGroup design also yields 1.27X higher throughput than NCCL when training ResNet-50 on the Summit system over 192 GPUs. The proposed designs have been made publicly available under MVAPICH2-GDR project [28]. As part of future work, we plan to evaluate the tree-based algorithm on top of the proposed NVGroup concept and perform a similar analysis for other collective operations at scale.

8 ACKNOWLEDGMENT

This research is supported in part by NSF grants #1931537, #1450440, #1664137, #1818253, and XRAC grant #NCR-130002.

The authors thank Drs. Olga Pearce and Kathryn Mohro for providing access to Lassen system. We also thank our colleague Mohammadreza Bayatpour for useful discussion and anonymous reviewers for the valuable comments.

REFERENCES

[1] AMD. 2016. ROCm Communication Collectives Library. https://github.com/ROCmSoftwarePlatform/rccl Accessed: May 1, 2020.

[2] Dario Amodei, Rishita Anubhai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Jingdong Chen, Mike Chrzanowski, Adam Coates, Greg Diamos, Erich Elsen, Jesse Engel, Linxi Fan, Christopher Fougner, Tony Han, Awni Y. Hannun, Billy Jun, Patrick LeGresley, Libby Lin, Sharan Narang, Andrew Y. Ng, Sherjil Ozair, Ryan Prenger, Jonathan Raiman, Sanjeev Satheesh, David Seetapun, Shubho Sengupta, Yi Wang, Zhiqian Wang, Chong Wang, Bo Xiao, Dani Yogatama, Jun Zhan, and Zhenyao Zhu. 2015. Deep Speech 2: End-to-End Speech Recognition in English and Mandarin. *CoRR* abs/1512.02595 (2015). arXiv:1512.02595 http://arxiv.org/abs/1512.02595

[3] Ammar Ahmad Awan, Jeroen Bédorf, Ching-Hsiang Chu, Hari Subramoni, and Dhabaleswar K. Panda. 2019. Scalable Distributed DNN Training using TensorFlow and CUDA-Aware MPI: Characterization, Designs, and Performance Evaluation. In *The 19th Annual IEEE/ACM International Symposium in Cluster, Cloud, and Grid Computing (CCGRID 2019)*.

[4] Ammar Ahmad Awan, Khaled Hamidouche, Jahanzeb Maqbool Hashmi, and Dhabaleswar K. Panda. 2017. 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 (PPoPP '17)*. ACM, New York, NY, USA, 193–205. https://doi.org/10.1145/3018743.3018769

[5] Ammar A. Awan, Arpan Jain, Ching-Hsiang Chu, Hari Subramoni, and Dhabaleswar K. Panda. 2020. Communication Profiling and Characterization of Deep-Learning Workloads on Clusters With High-Performance Interconnects. *IEEE Micro* 40, 1 (Jan 2020), 35–43. https://doi.org/10.1109/MM.2019.2949986

[6] Mohammadreza Bayatpour, Jahanzeb Maqbool Hashmi, Sourav Chakraborty, Hari Subramoni, Robert S Oakes, and Dhabaleswar K. Panda. 2018. SALaR: Scalable and Adaptive Designs for Large Message Reduction Collectives. (2018), 12–23.

[7] Tal Ben-Nun and Torsten Hoefler. 2019. Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis. *ACM Comput. Surv.* 52, 4, Article Article 65 (Aug. 2019), 43 pages. https://doi.org/10.1145/3320060

[8] F. Broquedis, J. Clet-Ortega, S. Moreaud, N. Furmento, B. Goglin, G. Mercier, S. Thibault, and R. Namyst. 2010. hwloc: A Generic Framework for Managing Hardware Affinities in HPC Applications. In *2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing*. 180–186.

[9] Z. Sura W. Hwu C. Pearson, I. Chung and J. Xiong. 2018. NUMA-aware Data-transfer Measurements for Power/NVLink Multi-GPU Systems,. In *International Workshop on OpenPOWER for HPC (IWOPHâĂŹ19) at the 2018 ISC High Performance Conference*.

[10] Minsik Cho, Ulrich Finkler, and David Kung. 2018. BlueConnect: Novel Hierarchical All-Reduce on Multi-tired Network for Deep Learning. In *32nd Conference on Neural Information Processing Systems (NeurIPS 2018)*.

[11] M. Cho, U. Finkler, M. Serrano, D. Kung, and H. Hunter. 2019. BlueConnect: Decomposing all-reduce for deep learning on heterogeneous network hierarchy. *IBM Journal of Research and Development* 63, 6 (2019), 1:1–1:11.

[12] Ching-Hsiang Chu, K. Hamidouche, A. Venkatesh, A. A. Awan, and D. K. Panda. 2016. CUDA Kernel Based Collective Reduction Operations on Large-scale GPU Clusters. In *2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*. 726–735. https://doi.org/10.1109/CCGrid.2016.111

[13] Ching-Hsiang Chu, Sreeram Potluri, Anshuman Goswami, Manjunath Gorentla Venkata, Neena Imam, and Chris J. Newburn. 2019. Designing High-Performance In-Memory Key-Value Operations with Persistent GPU Kernels and OpenSHMEM. In *OpenSHMEM and Related Technologies. OpenSHMEM in the Era of Extreme Heterogeneity*. Springer International Publishing, Cham, 148–164.

[14] Z. Dong, Y. L. Fang, X. Huang, H. Yan, S. Ha, W. Xu, Y. S. Chu, S. I. Campbell, and M. Lin. 2018. High-Performance Multi-Mode Ptychography Reconstruction on Distributed GPUs. In *2018 New York Scientific Data Summit (NYSDS)*. 1–5.

[15] Denis Foley and John Danskin. 2017. Ultra-Performance Pascal GPU and NVLink Interconnect. *IEEE Micro* 37, 2 (March 2017), 7–17. https://doi.org/10.1109/MM.2017.37

[16] Younghwan Go, Muhammad Asim Jamshed, YoungGyoun Moon, Changho Hwang, and KyoungSoo Park. 2017. APUNet: Revitalizing GPU as Packet Processing Accelerator. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. USENIX Association, Boston, MA, 83–96. https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/go

[17] K. Gupta, J. A. Stuart, and J. D. Owens. 2012. A study of Persistent Threads style GPU programming for GPGPU workloads. In *2012 Innovative Parallel Computing (InPar)*. 1–14. https://doi.org/10.1109/InPar.2012.6339596

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 770–778.

[19] IBM. 2018. IBM Spectrum MPI: Accelerating high-performance application parallelization. https://www.ibm.com/us-en/marketplace/spectrum-mpi. Accessed: May 1, 2020.

[20] Kawthar Shafie Khorassani, Ching-Hsiang Chu, Hari Subramoni, and Dhabaleswar K. Panda. 2018. Performance Evaluation of MPI Libraries on GPU-enabled OpenPOWER Architectures: Early Experiences. In *International Workshop on OpenPOWER for HPC (IWOPH 19) at the 2019 ISC High Performance Conference*.

[21] Pouya Kousha, Bharath Ramesh, Kaushik Kandadi Suresh, Ching-Hsiang Chu, Arpan Jain, Nick Sarkauskas, Hari Subramoni, and Dhabaleswar K. Panda. 2019. Designing a Profiling and Visualization Tool for Scalable and In-depth Analysis of High-Performance GPU Clusters. In *2019 IEEE 26th International Conference on High Performance Computing, Data, and Analytics (HiPC)*. 93–102.

[22] Nouamane Laanait, Joshua Romero, Junqi Yin, M. Todd Young, Sean Treichler, Vitalii Starchenko, Albina Borisevich, Alex Sergeev, and Michael Matheson. 2019. Exascale Deep Learning for Scientific Inverse Problems. arXiv:cs.LG/1909.11150

[23] Lawrence Livermore National Laboratory. 2018. Lassen | High Performance Computing. https://hpc.llnl.gov/hardware/platforms/lassen. Accessed: May 1, 2020.

[24] Xi Luo, Wei Wu, George Bosilca, Thananon Patinyasakdikul, Linnan Wang, and Jack Dongarra. 2018. ADAPT: An Event-based Adaptive Collective Communication Framework. In *Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing (HPDC '18)*. ACM, New York, NY, USA, 118–130. https://doi.org/10.1145/3208040.3208054

[25] Mark Harris and Kyrylo Perelygin . 2017. Cooperative Groups: Flexible CUDA Thread Programming. https://devblogs.nvidia.com/cooperative-groups/ Accessed: May 1, 2020.

[26] Message Passing Interface Forum. 2014. http://www.mpi-forum.org/ Accessed: May 1, 2020.

[27] S. A. Mojumder, M. S. Louis, Y. Sun, A. K. Ziabari, J. L. AbellÃăn, J. Kim, D. Kaeli, and A. Joshi. 2018. Profiling DNN Workloads on a Volta-based DGX-1 System. In *2018 IEEE International Symposium on Workload Characterization (IISWC)*. 122–133. https://doi.org/10.1109/IISWC.2018.8573521

[28] Network-Based Computing Laboratory. 2001. MVAPICH: MPI over InfiniBand, Omni-Path, Ethernet/iWARP, and RoCE. http://mvapich.cse.ohio-state.edu/. Accessed: May 1, 2020.

[29] NVIDIA. 2005. NVIDIA Management Library (NVML). https://developer.nvidia.com/nvidia-management-library-nvml Accessed: May 1, 2020.

[30] NVIDIA. 2011. NVIDIA GPUDirect. https://developer.nvidia.com/gpudirect Accessed: May 1, 2020.

[31] NVIDIA. 2016. NCCL. https://github.com/NVIDIA/nccl. Accessed: May 1, 2020.

[32] NVIDIA. 2017. NCCL Tests. https://github.com/NVIDIA/nccl-tests Accessed: May 1, 2020.

[33] Oak Ridge National Laboratory. 2018. Summit: America's Newest and Smartest Supercomputer. https://www.olcf.ornl.gov/summit/. Accessed: May 1, 2020.

[34] L. Oden, B. Klenk, and H. Froning. 2014. Energy-Efficient Collective Reduce and Allreduce Operations on Distributed GPUs. In *Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on*. 483–492.

[35] Open MPI. 2004. Open MPI: Open Source High Performance Computing. https://www.open-mpi.org/. Accessed: May 1, 2020.

[36] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.). Curran Associates, Inc., 8024–8035. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[37] Pitch Patarasuk and Xin Yuan. 2009. Bandwidth optimal all-reduce algorithms for clusters of workstations. *J. Parallel and Distrib. Comput.* 69, 2 (2009), 117 – 124. https://doi.org/10.1016/j.jpdc.2008.09.002

[38] Sreeram Potluri, Khaled Hamidouche, Akshay Venkatesh, Devendar Bureddy, and Dhabaleswar K Panda. 2013. Efficient Inter-node MPI Communication Using GPUDirect RDMA for InfiniBand Clusters With NVIDIA GPUs. In *Parallel Processing (ICPP), 2013 42nd International Conference on*. IEEE, 80–89.

[39] Rolf Rabenseifner. 2004. Optimization of Collective Reduction Operations. In *Computational Science - ICCS 2004*, Marian Bubak, Geert Dick van Albada, Peter M. A. Sloot, and Jack Dongarra (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 1–9.

[40] Peter Sanders, Jochen Speck, and Jesper Larsson TrÄdff. 2009. Two-tree algorithms for full bandwidth broadcast, reduction and scan. *Parallel Comput.* 35, 12 (2009), 581 – 594. https://doi.org/10.1016/j.parco.2009.09.001 Selected papers from the 14th European PVM/MPI Users Group Meeting.

[41] Jürgen Schmidhuber. 2015. Deep Learning in Neural Networks: An Overview. *Neural networks* 61 (2015), 85–117.

[42] Alexander Sergeev and Mike Del Balso. 2018. Horovod: fast and easy distributed deep learning in TensorFlow. *CoRR* abs/1802.05799 (2018). arXiv:1802.05799 http://arxiv.org/abs/1802.05799

[43] R. Shi, S. Potluri, K. Hamidouche, J. Perkins, M. Li, D. Rossetti, and D. K. Panda. 2014. 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)*. 1–10.

[44] Erich Strohmaier, Jack Dongarra, Horst Simon, and Martin Meuer. 1993. TOP 500 Supercomputer Sites. http://www.top500.org. Accessed: May 1, 2020.

[45] Sylvain Jeaugey. 2019. Massively Scale Your Deep Learning Training with NCCL 2.4. https://devblogs.nvidia.com/massively-scale-deep-learning-training-nccl-2-4/. Accessed: May 1, 2020.

[46] Rajeev Thakur, Rolf Rabenseifner, and William Gropp. 2005. Optimization of Collective Communication Operations in MPICH. *Int. J. High Perform. Comput. Appl.* 19, 1 (Feb. 2005), 49–66. https://doi.org/10.1177/1094342005051521

[47] T. Thao Nguyen, M. Wahib, and R. Takano. 2018. Hierarchical Distributed-Memory Multi-Leader MPI-Allreduce for Deep Learning Workloads. In *2018 Sixth International Symposium on Computing and Networking Workshops (CANDARW)*. 216–222. https://doi.org/10.1109/CANDARW.2018.00048

[48] Tiffany Trader. 2017. TSUBAME3.0 Points to Future HPE Pascal-NVLink-OPA Server. https://www.hpcwire.com/2017/02/17/tsubame3-0-points-future-hpe-pascal-nvlink-opa-server/ Accessed: May 1, 2020.

[49] Y. Ueno and R. Yokota. 2019. Exhaustive Study of Hierarchical AllReduce Patterns for Large Messages Between GPUs. In *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. 430–439.

[50] Guanhua Wangt, Amar Phanishayee, Shivaram Venkataraman, and Ion Stoicat. 2018. Blink: A fast NVLink-based collective communication library. (2018). https://rise.cs.berkeley.edu/wp-content/uploads/2018/01/blink-2-page-11_50.pdf Accessed: May 1, 2020.

[51] Udayanga Wickramasinghe and Andrew Lumsdaine. 2016. A Survey of Methods for Collective Communication Optimization and Tuning. *CoRR* abs/1611.06334 (2016). arXiv:1611.06334 http://arxiv.org/abs/1611.06334

[52] Hao Zhu, David Goodell, William Gropp, and Rajeev Thakur. 2009. Hierarchical Collectives in MPICH2. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, Matti Ropo, Jan Westerholm, and Jack Dongarra (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 325–326.