OmNICCL: Zero-cost Sparse AllReduce with Direct Cache Access and SmartNICs

Tongzhou Gu
KAUST

Jiawei Fei
NUDT

Marco Canini
KAUST

Abstract

AllReduce is a collective communication pattern commonly used in Distributed Deep Learning (DDL) and High Performance Computing (HPC). Sparse AllReduce, which compresses the data transmitted, achieves significant acceleration on specific workloads. However, compression introduces a non-negligible performance overhead. Therefore, we propose the OmNICreduce algorithm, an efficient inter-node sparse AllReduce method, as well as its implementation, OmNICCL. It utilizes Direct Cache Access (DCA) to achieve zero-overhead lossless compression and employs SmartNICs for aggregation on the data plane. We demonstrate that our method can provide up to a 7.24× speedup over conventional dense AllReduce methods under a 100Gbps RoCEv2 network and 1.76-17.37× performance improvement over state-of-the-art implementations when performing sparse AllReduce.

CCS Concepts

• Networks → In-network processing; **• Computer systems organization → Distributed architectures**.

Keywords

Collective Communication, In-Network Aggregation, DCA, DPU, SmartNIC

ACM Reference Format:
Tongzhou Gu, Jiawei Fei, and Marco Canini. 2024. OmNICCL: Zero-cost Sparse AllReduce with Direct Cache Access and SmartNICs. In *SIGCOMM Workshop on Networks for AI Computing (NAIC '24), August 4–8, 2024, Sydney, NSW, Australia.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3672198.3673804

1 Introduction

In the HPC area, OpenMP and MPI [1, 11, 35] are the most popular libraries for intra-node multithreading and inter-node multiprocessing, respectively. A survey [18] shows that almost all MPI programs rely on collective operations, among which AllReduce is the most commonly used. In the Machine Learning (ML) area, Parameter Servers [17, 27] and AllReduce [20] are two common methods to scale ML workloads across multiple nodes. With the rise of Large Language Models (LLM) in recent years, many LLM systems are dedicated to supporting model training at extreme scales [43, 46], with ReduceScatter or AllReduce being the underlying inter-node communication primitives. Therefore, it can be said that AllReduce is an essential building block for both HPC and ML workloads, affecting the entire system's performance.

Over the past three decades, many efforts have been put into accelerating AllReduce. The current de-facto standard implementations of AllReduce are MPI [1, 11, 35] and NCCL [31], which are mainly used for distributed CPU and GPU applications, respectively. They have implemented a series of dense AllReduce algorithms, including Tree [37] and Ring AllReduce [36], and adaptively choose more efficient algorithms based on the topology of intra- and inter-node connections and input size. They also support various transport methods, including shared memory, PCIe P2P, NVLink, TCP, and RDMA.

Hardware acceleration of AllReduce is also a viable approach. In-Network Aggregation (INA) is a method of AllReduce that uses additional hardware, typically switches, as aggregators. All workers usually stream data blocks to aggregators for aggregation, and then the aggregators broadcast the aggregated blocks to the workers. In large-scale networks, especially fat-trees, hierarchical AllReduce can also be performed between switches at different tiers. This method requires fewer communication hops than Ring AllReduce, reduces the computation of workers, and may provide greater overall bandwidth. Therefore, this method has lower latency, higher throughput, and better scalability. Previously, this method was implemented on commodity switches [14, 16, 21], programmable switches [19, 22, 40], and FPGAs [9, 12, 25, 28].

SmartNICs are a type of programmable network device that has emerged in recent years. They are equipped with additional processors, or expose the programming interface of existing NIC cores, allowing users to execute additional processing logic on the NIC or support more features. Accelerating collective communication with SmartNICs is also an active topic. Previous work [35] mainly utilized the feature of SmartNICs to initiate communication on behalf of hosts, offloading some work on the control plane to SmartNICs, freeing the CPU from these tasks, thereby reducing the latency of collectives and improving the overlap ratio of communication and computation of nonblocking collectives. Unlike INA, these works did not let SmartNICs handle data on the data plane, mainly considering the limited performance of SmartNICs, especially memory bandwidth and CPU performance.

Sparse AllReduce [9, 10, 38, 42] is another potential optimization opportunity. It reduces the amount of inter-node communication by compressing the transmitted data, either lossy or lossless. However, compression itself may also produce a noticeable performance overhead, especially when adopting the scheme of compressing first and then transmitting, that is, the network is idle during compression. This often makes sparse AllReduce not only fail to reduce communication when aggregating dense arrays, but also consume the same time as dense AllReduce on communication, and consume more time on compression.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
NAIC '24, August 4–8, 2024, Sydney, NSW, Australia
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0713-1/24/08
https://doi.org/10.1145/3672198.3673804

Therefore, we propose OmNICreduce, an efficient inter-node sparse INA algorithm, as well as its implementation OmNICCL. It uses NVIDIA BlueField-2 SmartNICs or CPU servers as aggregators to perform aggregation on the data plane of a 100Gbps RDMA network, and uses DCA and Kernel Fusion to eliminate memory access of inlined decompression and aggregation, solving the problem of insufficient memory bandwidth of SmartNICs. For CPU workers, it also uses DCA and Kernel Fusion to eliminate memory access of inlined compression and batching; it uses pipelining to overlap compression and communication to mask the compression latency; and it achieves zero-cost lossless compression. For GPU workers, it uses GPUDirect RDMA (GDR) and Scatter Gather DMA to batch data blocks, while avoiding frequent GPU-CPU communication.

OmNICreduce and its realization in OmNICCL achieve all of the following goals:

- For dense input arrays, it provides performance similar to or better than dense AllReduce libraries. For sparse input arrays, it provides performance superior to the state-of-the-art sparse INA.
- It enables SmartNIC-based aggregators to provide performance similar to server-based aggregators.
- It supports efficient aggregation of arrays on both CPUs and GPUs.

We demonstrate that our method, when using BlueField-2 aggregators for AllReduce, achieves up to a 7.24× performance improvement over various MPI implementations for dense CPU arrays. For dense GPU arrays, it provides performance similar to NCCL. For sparse arrays, our method outperforms OmniReduce [10], a state-of-the-art sparse INA implementation, achieving up to 2.96× and 17.37× performance improvements on CPU and GPU, respectively. We also verify that BlueField-2 aggregators can deliver performance similar to server-based aggregators.

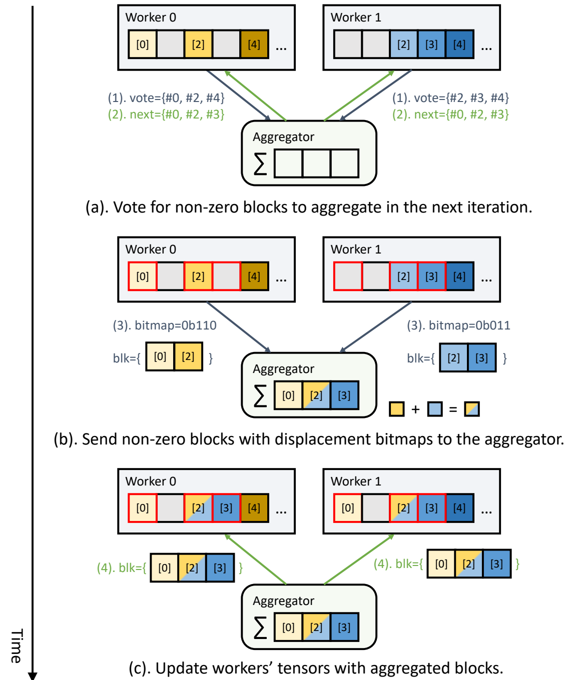

Figure 1: Overview of the OmNICreduce algorithm. The grey blocks represent zero blocks, while the non-grey blocks are non-zero blocks. The text next to the arrows indicates the messages between the worker and the aggregator. The sequence number of the message is indicated by the number in the text, and the direction of transmission corresponds to the direction of the arrow with the same color.

2 Design

This section presents the design of OmNICreduce, which adopts the core idea of OmniReduce, as well as its implementation OmNICCL. The input array is divided, or rather viewed, as a series of blocks of size b that are continuously arranged in memory. These blocks are categorized into zero blocks and non-zero blocks, where zero blocks contain only zeros and non-zero blocks contain at least one non-zero value. Workers only send non-zero blocks to aggregators, skipping the transmission of zero blocks. This simple lossless compression method has been proven to accelerate various deep learning models significantly [10, 42]. We adopt this method mainly because it is simple enough to be combined with DCA and Kernel Fusion to achieve zero-cost lossless compression.

2.1 OmNICreduce Algorithm

Figure 1a illustrates the first step of the algorithm, voting, where workers vote to select the blocks to be aggregated in the next iteration. Workers first send the indices of the next d non-zero blocks of local input arrays to aggregators. Then, aggregators find the smallest top-d indices and return them to workers as global indices. These global indices of the blocks will be aggregated and updated in the next iteration.

Figure 1b shows the second step of the algorithm, batch sending with compression. Workers receive global indices in the previous voting step and need to send the non-zero blocks pointed to by these indices to aggregators. However, the indices may point to zero blocks on some workers, such as Block #3 on Worker 0, which is a zero block. Therefore, workers also generate displacement bitmaps, where each bit of the bitmap represents whether the corresponding global index points to a zero block that is skipped for transmission. In the actual implementation, workers reorganize non-zero blocks as the payload of a single RDMA message for transmission. Aggregators do not record each worker's vote to save memory, but rely on the bitmap to decompress the message. Saving memory consumption is the core of this work, and the reason for doing so will be explained in Section 2.2.

Figure 1c depicts the third step of the algorithm, updating. In each iteration, as soon as a worker's message is received, aggregators immediately add the payload of the message to the partial-aggregated blocks in memory. When aggregators receive messages from all workers, the aggregator sends the fully-aggregated blocks to workers, which then update the corresponding blocks according to the global indices.

In the actual implementation, the above three steps are carried out simultaneously. That is, the message sent by the worker to

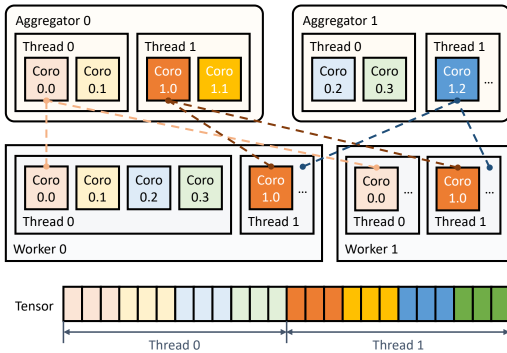

Figure 2: Parallelism method of OmNICCL. Coro stands for coroutine. Each coroutine is an instance of the algorithm, and coroutines with the same number perform aggregation among themselves. The diagram below shows the way the array is partitioned, where coroutines are responsible for aggregating blocks of the same color.

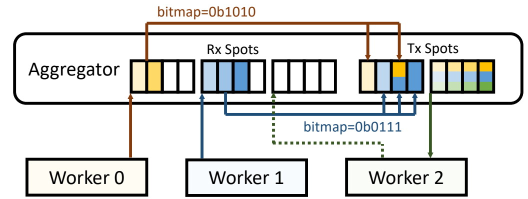

Figure 3: Memory layout of the aggregator and the data movement path of the entire system. Worker 0 and Worker 1 are in the Send phase of the current iteration, while Worker 2 is in the Update phase of the previous iteration.

the aggregator contains the local indices of the next iteration, the displacement bitmap of the current iteration, and the non-zero blocks. The message sent by the aggregator to the worker contains the global indices of the next iteration and the aggregated blocks. Among them, the bitmap is embedded in the RDMA immediate, and the blocks and indices are combined as the payload.

2.1.1 Parallelism. To maximize the throughput of the system, we extend it to multiple aggregators and hardware threads. We assume that in the real world, each worker is equipped with a SmartNIC as an aggregator. As shown in Figure 2, OmNICCL splits the array into multiple sub-arrays and performs AllReduce on the sub-arrays in parallel, i.e., it splits an AllReduce task into multiple sub-tasks. The sub-tasks are dispatched to different software coroutines. Each software coroutine runs an instance of the algorithm, and the coroutine of the worker talks to the corresponding coroutine of the aggregator. Multiple coroutines run on each hardware thread. This parallelization method can increase the total achievable network bandwidth of the aggregators and overlap computation and communication.

2.1.2 Block Merge. OmNICreduce is designed so as to aggregate multiple blocks at once, a feature we call Block Merge. It can be noted that a single RDMA message from the workers of OmNICreduce contains d non-zero blocks (at most). This is because we want to reduce the block size b to better exploit data sparsity, while increasing the message size db to improve the utilization of network bandwidth. This idea is similar to Block Fusion of OmniReduce. However, the difference is that Block Fusion reassembles the messages sent by multiple algorithm instances, while a single algorithm instance of Block Merge can produce a merged message.

Assume that there are n workers in the system. For a single instance of the OmniReduce aggregator algorithm, it records the local indices of each worker, resulting in a space complexity of $O(n)$. Each time the algorithm instance receives a message from a worker, it needs to perform $O(\log n)$ computations to find the smallest local index, thereby determining whether the iteration has ended. When Block Fusion is enabled, it creates multiple algorithm instances and fuses the messages of these instances, thus generating a space and time complexity of $O(dn)$ and $O(d \log n)$, respectively. In contrast, Block Merge does not need to create multiple algorithm instances or record local indices. Its algorithm instance only requires $O(d)$ space to store the top-d smallest global indices. Each time it receives a message from a worker, it only needs to perform a merge operation with a time complexity of $O(d)$ to calculate the top-d smallest global indices. It uses only one counter to determine whether the iteration has ended. Therefore, we conclude that Block Merge has superior algorithmic time and space complexity compared to Block Fusion.

2.2 OmNICCL Aggregator

In OmNICreduce algorithm, the aggregator is responsible for aggregating non-zero blocks in a streaming manner, while calculating the global indices. There are several challenges to implement OmNICreduce aggregator onto the SmartNIC.

2.2.1 Challenge 1: Limited memory bandwidth of SmartNICs. We aim to use SmartNICs as aggregators, but due to considerations of cost, energy efficiency, and design objectives, the performance of SmartNICs is often limited. Taking BlueField-2, which is mainly used in this work, as an example, we tested it with STREAM Benchmark [29] and found that BlueField-2 can only provide 83.6Gbps memory bandwidth, which is far from enough to support bidirectional 200Gbps network transmission.

2.2.2 Solution 1: Remove memory accesses with DCA. Interestingly, we found that BlueField-2 seems to support some form of DCA, which is also the reason why it can achieve bidirectional 200Gbps throughput in the OFED Perftest [4] `ib_write_bw` test. Therefore, we hope to use DCA to bypass the problem of insufficient memory bandwidth and thus accelerate the algorithm of the aggregator.

Ideally, DCA allows the NIC to read and write directly to the Last Level Cache (LLC) without generating any memory transactions. This requires the LLC to have enough space to store the data received by the NIC, and the data to be sent resides in the LLC without being evicted. The most direct way to achieve this is to save as much program memory as possible so that the program memory is fully present in the cache. The Appendix provides additional background on DCA.

2.2.3 Challenge 2: Limited LLC capacity on SmartNICs. In order to make full use of DCA, the cache usage on the SmartNIC requires special attention. However, The LLC capacity on SmartNICs tends

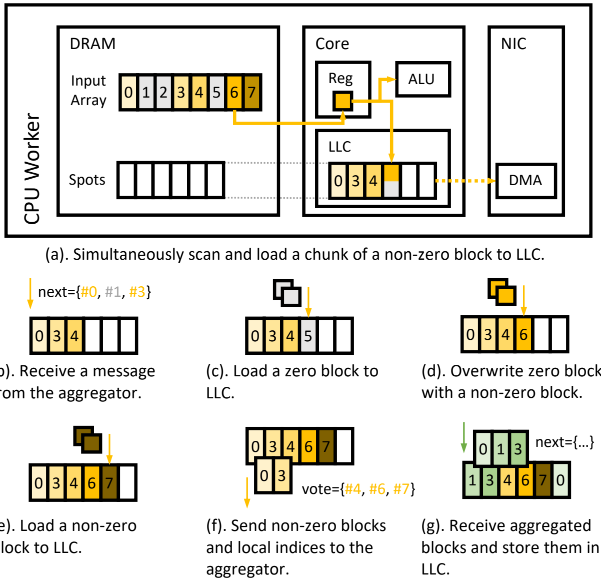

Figure 4: Memory layout of the worker and lifecycle of the blocks in the ring buffer. Yellow represents non-zero blocks, grey represents zero blocks, and green represents aggregated blocks. Blocks #0, #3, and #4 were loaded in the previous iteration. Among them, #0 and #3 will be sent and overwritten by aggregated blocks, while #6 and #7 are loaded into the buffer for the next iteration.

to be extremely limited. In our case, there is only 6MB LLC on BlueField-2.

2.2.4 Solution 2: Efficient memory scheduling. Figure 3 shows the memory layout of OmNICCL aggregator. A Spot is a part of an RDMA Memory Region (MR), each Spot is enough to store d blocks and indices. Each worker has its dedicated Rx Spot on the aggregator for receiving non-zero blocks and local indices. Meanwhile, the aggregator needs two Tx Spots, one for storing the partial aggregated blocks of the current iteration, and one for storing the fully aggregated blocks of the previous iteration, forming a double buffering.

Whenever the aggregator receives a message from the worker, it immediately adds the non-zero blocks in the Rx Spot to the partial aggregated blocks in the Tx Spot according to the displacement bitmap, without the need for extra memory to store the decompressed data. The reason for using double buffering is that there may be workers that have not completed the transmission of the previous iteration, such as Worker 2 in Figure 3, so the Tx Spot in transmission cannot be directly overwritten. Therefore, each algorithm instance of the aggregator only consumes about $(n+2)bd$ memory, where n is the number of workers; to get reasonable performance, usually block size b is taken as 1KB, merge depth d is taken as 16. Theoretically, BlueField-2's 6MB LLC can support the aggregation of more than 300 workers.

2.3 OmNICCL CPU Worker

There are also challenges to the implementation of OmNICreduce's worker component.

2.3.1 Challenge 3: Overheads of non-zero block scanning and batching are non-negligible. CPU workers need to first scan the entire array to find all non-zero blocks before they can start sending them to aggregators. Scanning the array is a memory-bound operation. Constrained by the CPU memory bandwidth or PCIe, this operation is time-consuming, regardless of whether it is performed directly by the CPU or by the GPU.

OmNICreduce, as an efficient block-based Sparse INA algorithm, is designed to assemble multiple blocks into a single message for transmission. This approach not only fully exploits sparsity but also enhances network bandwidth utilization. However, the reassembly of blocks may potentially introduce performance overhead.

2.3.2 Solution 3: Zero-cost compression and batching. Therefore, we consider sending non-zero blocks while scanning. Intuitively, this scanning operation seems to inevitably bring some overhead. Here, we propose a zero-cost lossless compression method that eliminates memory access caused by scanning with the help of DCA. As shown in Figure 4a, each worker algorithm instance creates a ring buffer that can hold $2d$ blocks and is registered as an RDMA MR, serving as a Tx/Rx Spot. This ring buffer is small in size and frequently accessed, so it can reside in the LLC. When the worker enters the next iteration (Figure 4b), the worker needs to send the non-zero blocks of the current iteration and the voting indices of the next iteration. In Figure 4b, Blocks #0 and #3 will be sent, and the worker needs to know the indices of the two non-zero blocks after #4. Therefore, in Figure 4c, the worker tries to check whether #5 is a non-zero block.

After the CPU reads the chunk of #5 from DRAM into the register, it uses the Arithmetic Logic Unit (ALU) to judge whether the chunk contains non-zero values while writing the chunk into the Spot. This is because RDMA cannot directly read and write the non-MR input array, so it needs to copy data into the Spot. However, since the Spot is in the LLC, it does not generate actual memory access, and the modern CPU microarchitecture can simultaneously read and write data and perform calculations. Note that until all values have been checked, we cannot determine whether a block is a non-zero block. If we check first and then copy the block, it may not only generate additional memory access but also waste CPU cycles.

However, #5 is a zero block, so in Figure 4d, the worker turns to check #6 and overwrites the previously read #5. In the process of scanning and reading multiple blocks (Figure 4e), the blocks incidentally complete reassembly, that is, zero-cost batching. The NIC then directly reads data from the LLC and sends it (Figure 4f); it then uses the free position of the Spot to receive data from the aggregator (Figure 4g), and finally uses the CPU to copy back to the destination address.

In this process, the worker only reads and writes the array once to complete the scanning, sending, and updating of blocks, and it does not block network transmission; and so, we refer to this method as zero-cost compression. In addition, this method does not require registering user arrays as MRs, thereby freeing OmNICCL from managing MRs.

2.4 OmNICCL GPU Worker

Unlike CPU workers, the GPU, with its much larger memory bandwidth than the CPU, makes the scanning time extremely short. Therefore, it is acceptable to scan first and then transmit, which is exactly what OmniReduce did. OmNICCL also follows this practice. Specifically, the GPU first scans and copies the input array into a large MR. The scanning procedure generates a bitmap, where each bit indicates whether the corresponding block is a non-zero block. This bitmap is sent back to the CPU, and then the CPU uses the bitmap to calculate the voting indices before initiating the transmission.

Different from OmniReduce, the scanning and copying in OmNICCL are fused into one kernel. In addition, OmNICCL uses RDMA Scatter/Gather List (SGL) to combine multiple blocks and indices located in the CPU into a single RDMA message. RDMA reads and writes the MR on the GPU directly through GDR, and finally, the GPU copies the aggregated array in the MR back to the destination address.

3 Evaluation

In this section, we present the preliminary evaluation of OmNICCL. We aim to answer the following questions:

- How much performance improvement does OmNICCL provide over industry standard dense AllReduce libraries and OmniReduce in terms of latency and throughput under different configurations (i.e., block size, sparsity, number of nodes)?
- Does the use of BlueField-2 aggregators introduce any performance degradation to OmNICCL?

Experimental setup. The specifications of our testbed and BlueField-2 are shown in the Appendix in Table 1 and Table 2, respectively.

For network configuration, we set up a 100Gbps RoCEv2 network. We used an Edgecore DCS810 switch, which is equipped with an Intel Tofino 2 programmable switch chip and supports up to 400Gbps speed. On the switch, we installed the EdgeCore SONiC OS and enabled L3 PFC and ECN mechanisms. On the NIC, we enabled L3 PFC and the DCQCN congestion control algorithm.

For network topology, all NICs are connected to the switch via 3 to 10-meter 100Gbps fibers or DAC cables, and nodes of the same type use the same fibers or cables. The port on BlueField-2 is dedicated to the BlueField SoC, allowing the BlueField SoC to operate in a manner similar to an independent server.

For machines with multiple NUMA nodes, we bind applications to the cores and NUMA nodes closest to the ConnectX NICs. For GPU machines, we ensure that the GPUs used support GDR. All machines run Ubuntu 22.04 (Linux 5.15), OFED 23.07, CUDA 12.2, while BlueField-2 runs Ubuntu 20.04 (Linux 5.4.0), OFED 5.7.

Results. We have implemented our own microbenchmark, similar to the OSU MPI Microbenchmark. It supports AllReduce benchmarking for OmNICCL, MPI, and NCCL with varying sparsity and array sizes. Currently, it generates identical arrays for different machines as input for AllReduce. For dense AllReduce, we have chosen NVIDIA HPC-X MPI, Intel MPI, and NCCL for comparison. Due to device resource limitations, our current testbed only has one BlueField-2 available. To evaluate the scalability of OmNICCL, we have constructed a multi-machine experimental environment

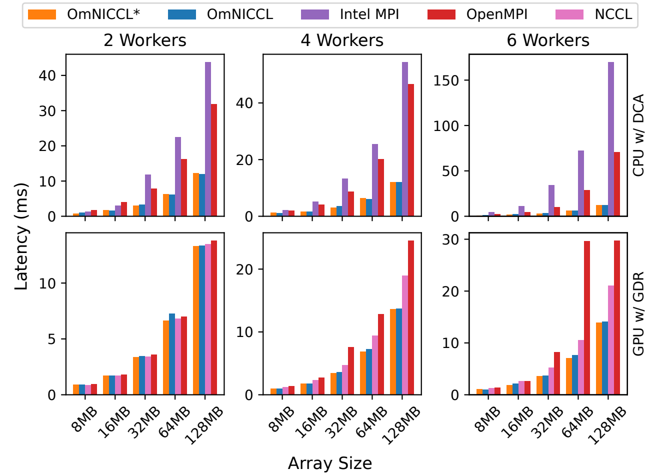

Figure 5: Latency of dense AllReduce. The asterisk represents the use of a BlueField-2 aggregator, while the rest are CPU aggregators. OmNICCL can achieve up to 7.24× and 1.52× speedup on AllReduce CPU and GPU arrays, respectively.

with a mix of CPU servers and SmartNICs as aggregators in the absence of BlueField-2 SmartNICs.

In this environment, we use $n-1$ Skylake CPU servers and a single BlueField-2 as aggregators, and n Haswell CPU / V100 GPU servers as workers for testing. From a performance evaluation perspective, this environment is equivalent to a multi-BlueField-2 NIC environment because OmNICCL evenly distributes sub-tasks to aggregators, and the system throughput is limited by the slowest aggregator. We also use n CPU servers as aggregators to evaluate whether BlueField-2 aggregators would negatively impact performance.

Figure 5 shows the performance comparison of OmNICCL, NCCL, and MPI performing AllReduce on CPU and GPU arrays under different numbers of workers. The microbenchmark generates a completely dense array of 8 to 128MB as input. We bind the program to the NUMA node with optimal NIC affinity, and run one MPI or OmNICCL process on each machine. For CPU arrays, OmNICCL achieves a speedup of 1.67-2.59×, 1.52-3.84×, and 2.31-7.24× compared to the optimal MPI implementation under 2, 4, and 6 workers, respectively. We believe this is because MPI always uses a single CPU core for aggregation, and for a single core, it is difficult to handle a large data stream. For GPU arrays, OmNICCL achieves 95% to 102% performance with 2 workers, and 1.23-1.39× and 1.24-1.52× speedup with 4 and 6 workers, respectively. We speculate that this is because when there are more than 2 workers, OmNICCL's larger overall bandwidth is helpful. In summary, OmNICCL can provide comparable or even superior performance to NCCL and MPI in completely dense situations. This is mainly because zero-cost compression with DCA and near-zero cost compression with GDR make the processing overhead of sparse data close to zero.

Figure 6 shows the Allreduce latency of OmNICCL under different sparsity and different numbers of workers. When the sparsity is small, OmNICCL can fully utilize the network bandwidth. As the sparsity increases, the Allreduce latency of OmNICCL continues to

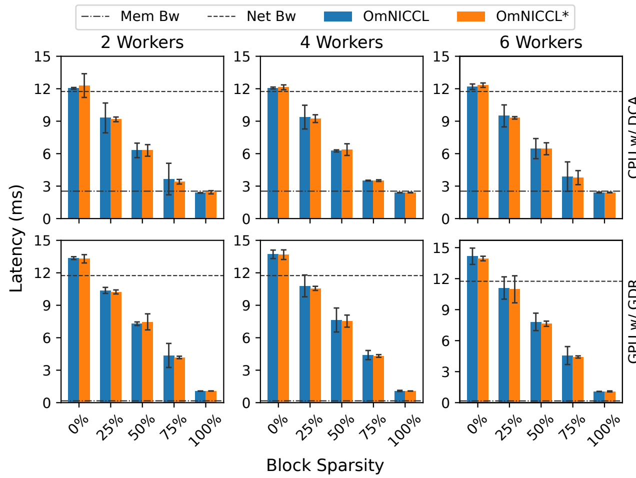

Figure 6: Latency of sparse AllReduce on 128MB array. The asterisk represents the use of a BlueField-2 aggregator. 'Mem Bw' and 'Net Bw' represent the latency of array size divided by memory and network bandwidth, respectively, which are the lower bounds of dense and sparse AllReduce latencies.

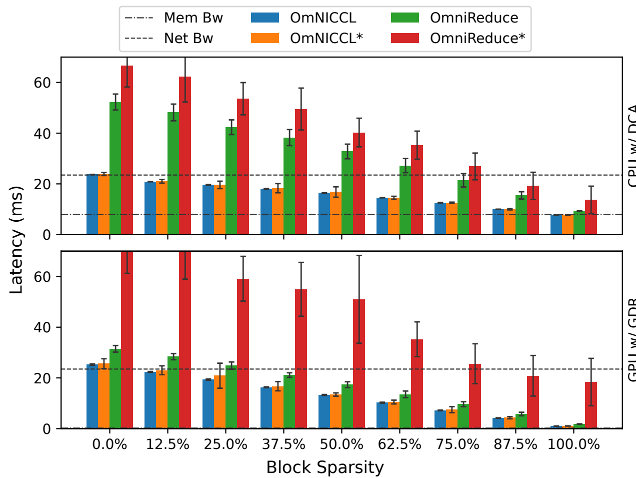

Figure 7: Latency of sparse AllReduce on 256MB array. The asterisk represents the use of a BlueField-2 aggregator. Employ a CPU / BlueField-2 aggregator and a CPU / GPU worker.

decrease, and is eventually constrained by memory bandwidth. On the current hardware platform, a sparsity of 75% brings a speedup of 3.11× and 3.30× to CPU workers and GPU workers compared to dense OmNICCL, respectively.

Figure 7 shows the latency comparison of OmNICCL and OmniReduce performing CPU and GPU AllReduce operations when a single BlueField / Skylake CPU server is used as an aggregator and a single A100 GPU / Skylake CPU server is used as a worker. The results show that whether for CPU arrays or GPU arrays, the performance of OmNICCL is superior to OmniReduce. We also find that the performance of OmniReduce's algorithm is greatly reduced when deployed to SmartNICs. OmNICCL is 1.76-2.96× and 2.83-17.37× faster than OmniReduce on CPU and GPU arrays, respectively, while the performance of OmNICCL deployed on SmartNICs and servers is basically the same. This is mainly due to the efficient algorithm and implementation of OmNICCL, which makes its hardware requirements much lower than OmniReduce, making it more suitable for SmartNICs with insufficient computing power and tight resources.

4 Related Work

Switch-based INA solutions including DAIET [39], SwitchML [40], ATP [19], P4COM [2], NetSHa [45], ESA [41] and NetEC [3] offload aggregation onto the dataplane of programmable switches. To overcome the limited switch resources and improve processing of large packet payloads, NetReduce [25], PANAMA [12], SwitchAgg [44], iSwitch [21] and AR-Switch [26] propose FPGA-based INA solutions. Even though FPGAs can support richer logic as compared to programmable switches, it remains unclear whether the above solutions can be deployed in practice since non-trivial cost considerations apply and the ease of programming FPGA remains a challenge. SHARP [14, 30] is the first commercial INA solution, which is based on the InfiniBand protocol and specialized network hardware.

Focusing on the optimization of sparse collective communication, Libra [34] designs a specialized INA solution for sparse models, which only aggregates gradients of hot parameters in programmable switch. FLARE [9] designs a new type of programmable switch prototype, which provides hardware-level support for sparse INA. OmniReduce [10] proposes an efficient sparse aggregation algorithm. However in order to deploy on programmable switches, the algorithm requires to be simplified.

SmartNICs provide an alternate avenue for implementation. Some works [7, 13] accelerate MPI collective operations by offloading operations onto a BlueField SmartNIC. However, unlike INA, these works do not take full advantage of the dataplane of SmartNICs. To the best of our knowledge, OmNICreduce is the first SmartNIC-based sparse INA solution which is compatible with RDMA and GDR.

5 Conclusion

Sparse AllReduce is an optimization opportunity for accelerating collective operations in the context of HPC and ML workloads. We presented the preliminary design of OmNICreduce, an efficient streaming algorithm for block-based inter-node sparse AllReduce, and the evaluation of its realization in OmNICCL, which leverages DCA to achieve zero-overhead lossless compression and uses a BlueField-2 for aggregation. We demonstrated that this approach is effective in leveraging scarce hardware resources, and accelerates previous implementations in both dense and sparse input scenarios.

Acknowledgments

This publication is based upon work supported by the King Abdullah University of Science and Technology (KAUST) Office of Research Administration (ORA) under Award No. ORA-CRG2020-4382. For computer time, this research used the resources of the Supercomputing Laboratory at KAUST.

References

[1] 1993. MPI: A message passing interface. In SC. https://doi.org/10.1145/169627.169855

[2] 2021. P4com: In-network computation with programmable switches. arXiv:2107.13694 [cs.NI] https://arxiv.org/abs/2107.13694

[3] 2022. NetEC: Accelerating erasure coding reconstruction with in-network aggregation, author=Qiao, Yi and Zhang, Menghao and Zhou, Yu and Kong, Xiao and Zhang, Han and Xu, Mingwei and Bi, Jun and Wang, Jilong. IEEE Transactions on Parallel and Distributed Systems 33, 10 (2022). https://doi.org/10.1109/TPDS.2022.3145836

[4] 2023. Open Fabrics Enterprise Distribution (OFED) Performance Tests. https://github.com/linux-rdma/perftest

[5] 2023. RDMA Core Userspace Libraries and Daemons. https://github.com/linux-rdma/rdma-core

[6] Arm. 2023. Arm DynamIQ Shared Unit Technical Reference Manual r3p0. https://developer.arm.com/documentation/100453/0300/functional-description/l3-cache/cache-stashing

[7] Mohammadreza Bayatpour, Nick Sarkauskas, Hari Subramoni, Jahanzeb Maqbool Hashmi, and Dhabaleswar K Panda. 2021. BluesMPI: Efficient MPI Non-blocking Alltoall Offloading Designs on Modern BlueField Smart NICs. In ISC. https://doi.org/10.1007/978-3-030-78713-4_2

[8] Jeffrey Brown, Sandra Woodward, Brian Bass, and Charlie Johnson. 2011. IBM Power Edge of Network Processor: A Wire-Speed System on a Chip. IEEE Micro (2011). https://doi.org/10.1109/MM.2011.3

[9] Daniele De Sensi, Salvatore Di Girolamo, Saleh Ashkboos, Shigang Li, and Torsten Hoefler. 2021. Flare: Flexible in-Network Allreduce. In SC. https://doi.org/10.1145/3458817.3476178

[10] Jiawei Fei, Chen-Yu Ho, Atal N. Sahu, Marco Canini, and Amedeo Sapio. 2021. Efficient Sparse Collective Communication and its application to Accelerate Distributed Deep Learning. In SIGCOMM. https://doi.org/10.1145/3452296.3472904

[11] Edgar Gabriel, Graham E. Fagg, George Bosilca, Thara Angskun, Jack J. Dongarra, Jeffrey M. Squyres, Vishal Sahay, Prabhanjan Kambadur, Brian Barrett, Andrew Lumsdaine, Ralph H. Castain, David J. Daniel, Richard L. Graham, and Timothy S. Woodall. 2004. Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation. In EuroPVM/MPI. https://doi.org/10.1007/978-3-540-30218-6_19

[12] Nadeen Gebara, Paolo Costa, and Manya Ghobadi. 2021. In-network Aggregation for Shared Machine Learning Clusters. In MLSys.

[13] Richard Graham, George Bosilca, Yong Qin, Bradley Settlemyer, Gilad Shainer, Craig Stunkel, Geoffroy Vallee, Brody Williams, Gerardo Cisneros-Stoianowski, Sebastian Ohlmann, and Markus Rampp. 2024. Optimizing Application Performance with BlueField: Accelerating Large-Message Blocking and Nonblocking Collective Operations. In ISC. https://doi.org/10.23919/ISC.2024.10528935

[14] Richard L. Graham, Devendar Bureddy, Pak Lui, Hal Rosenstock, Gilad Shainer, Gil Bloch, Dror Goldenerg, Mike Dubman, Sasha Kotchubievsky, Vladimir Koushnir, Lion Levi, Alex Margolin, Tamir Ronen, Alexander Shpiner, Oded Wertheim, and Eitan Zahavi. 2016. Scalable Hierarchical Aggregation Protocol (SHArP): A Hardware Architecture for Efficient Data Reduction. In COMHPC. https://doi.org/10.1109/COMHPC.2016.006

[15] Intel. 2023. Intel Data Direct I/O Technology (Intel DDIO): Technology Brief. https://www.intel.com/content/www/us/en/io/data-direct-i-o-technology-brief.html

[16] Alexander Ishii and Ryan Wells. 2022. The Nvlink-Network Switch: Nvidia's Switch Chip for High Communication-Bandwidth Superpods. In HCS. https://doi.org/10.1109/HCS55958.2022.9895480

[17] Yimin Jiang, Yibo Zhu, Chang Lan, Bairen Yi, Yong Cui, and Chuanxiong Guo. 2020. A Unified Architecture for Accelerating Distributed DNN Training in Heterogeneous GPU/CPU Clusters. In OSDI. https://www.usenix.org/conference/osdi20/presentation/jiang

[18] Ignacio Laguna, Ryan Marshall, Kathryn Mohror, Martin Ruefenacht, Anthony Skjellum, and Nawrin Sultana. 2019. A large-scale study of MPI usage in open-source HPC applications. In SC. https://doi.org/10.1145/3295500.3356176

[19] ChonLam Lao, Yanfang Le, Kshiteej Mahajan, Yixi Chen, Wenfei Wu, Aditya Akella, and Michael Swift. 2021. ATP: In-network Aggregation for Multi-tenant Learning. In NSDI. https://www.usenix.org/conference/nsdi21/presentation/lao

[20] Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, and Soumith Chintala. 2020. PyTorch Distributed: Experiences on Accelerating Data Parallel Training. Proc. VLDB Endow. (2020). https://doi.org/10.14778/3415478.3415530

[21] Youjie Li, Iou-Jen Liu, Yifan Yuan, Deming Chen, Alexander Schwing, and Jian Huang. 2019. Accelerating Distributed Reinforcement learning with In-Switch Computing. In ISCA. https://doi.org/10.1109/ISCA.2019.00034

[22] Zhaoyi Li, Jiawei Huang, Yijun Li, Aikun Xu, Shengwen Zhou, Jingling Liu, and Jianxin Wang. 2023. A2TP: Aggregator-Aware In-Network Aggregation for Multi-Tenant Learning. In EuroSys. https://doi.org/10.1145/3552326.3587436

[23] Linux Foundation. 2015. Data Plane Development Kit (DPDK). http://www.dpdk.org

[24] Ming Liu, Tianyi Cui, Henry Schuh, Arvind Krishnamurthy, Simon Peter, and Karan Gupta. 2019. Offloading Distributed Applications onto SmartNICs Using IPipe. In SIGCOMM. https://doi.org/10.1145/3341302.3342079

[25] Shuo Liu, Qiaoling Wang, Junyi Zhang, Wenfei Wu, Qinliang Lin, Yao Liu, Meng Xu, Marco Canini, Ray C. C. Cheung, and Jianfei He. 2023. In-Network Aggregation with Transport Transparency for Distributed Training. In ASPLOS. https://doi.org/10.1145/3582016.3582037

[26] Yao Liu, Junyi Zhang, Shuo Liu, Qiaoling Wang, Wangchen Dai, and Ray Chak Chung Cheung. 2021. Scalable Fully Pipelined Hardware Architecture for In-Network Aggregated AllReduce Communication. IEEE Transactions on Circuits and Systems I: Regular Papers 68, 10 (2021). https://doi.org/10.1109/TCSI.2021.3098841

[27] Liang Luo, Jacob Nelson, Luis Ceze, Amar Phanishayee, and Arvind Krishnamurthy. 2018. Parameter Hub: A Rack-Scale Parameter Server for Distributed Deep Neural Network Training. In SoCC. https://doi.org/10.1145/3267809.3267840

[28] Luo Mai, Lukas Rupprecht, Abdul Alim, Paolo Costa, Matteo Migliavacca, Peter Pietzuch, and Alexander L. Wolf. 2014. NetAgg: Using Middleboxes for Application-Specific On-Path Aggregation in Data Centres. In CoNEXT. https://doi.org/10.1145/2674005.2674996

[29] John D. McCalpin. 1991-2007. STREAM: Sustainable Memory Bandwidth in High Performance Computers. http://www.cs.virginia.edu/stream/

[30] Mellanox. 2021. Scalable Hierarchical Aggregation and Reduction Protocol (SHARP). https://www.mellanox.com/products/sharp.

[31] Nvidia. 2015. Nvidia Collective Communications Library (NCCL). https://developer.nvidia.com/nccl

[32] Nvidia. 2023. Nvidia BlueField-2 DPU. https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/documents/datasheet-nvidia-bluefield-2-dpu.pdf

[33] Nvidia. 2023. Nvidia BlueField-3 DPU. https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/documents/datasheet-nvidia-bluefield-3-dpu.pdf

[34] Heng Pan, Penglai Cui, Zhenyu li, Ru Jia, Penghao Zhang, Leilei Zhang, Ye Yang, Jiahao Wu, Jianbo Dong, Zheng Cao, Qiang Li, Hongqiang Harry Liu, Mathy Laurent, and Gaogang Xie. 2022. Enabling Fast and Flexible Distributed Deep Learning with Programmable Switches. arXiv:2205.05243 [cs.NI] https://arxiv.org/abs/2205.05243

[35] Dhabaleswar Kumar Panda, Hari Subramoni, Ching-Hsiang Chu, and Mohammadreza Bayatpour. 2021. The MVAPICH project: Transforming research into high-performance MPI library for HPC community. Journal of Computational Science (2021). https://doi.org/10.1016/j.jocs.2020.101208

[36] Pitch Patarasuk and Xin Yuan. 2009. Bandwidth optimal all-reduce algorithms for clusters of workstations. J. Parallel Distributed Comput. (2009). https://doi.org/10.1016/j.jpdc.2008.08.008

[37] Rolf Rabenseifner. 2004. Optimization of Collective Reduction Operations. In ICCS. https://doi.org/10.1007/978-3-540-24685-5_1

[38] Cedric Renggli, Saleh Ashkboos, Mehdi Aghagolzadeh, Dan Alistarh, and Torsten Hoefler. 2019. SparCML: High-Performance Sparse Communication for Machine Learning. In SC. https://doi.org/10.1145/3295500.3356222

[39] Amedeo Sapio, Ibrahim Abdelaziz, Abdulla Aldilaijan, Marco Canini, and Panos Kalnis. 2017. In-Network Computation is a Dumb Idea Whose Time Has Come. In HotNets. https://doi.org/10.1145/3152434.3152461

[40] Amedeo Sapio, Marco Canini, Chen-Yu Ho, Jacob Nelson, Panos Kalnis, Changhoon Kim, Arvind Krishnamurthy, Masoud Moshref, Dan Ports, and Peter Richtarik. 2021. Scaling Distributed Machine Learning with In-Network Aggregation. In NSDI. https://www.usenix.org/conference/nsdi21/presentation/sapio

[41] Hao Wang, Yuxuan Qin, ChonLam Lao, Yanfang Le, Wenfei Wu, and Kai Chen. 2022. Efficient data-plane memory scheduling for in-network aggregation. arXiv:2201.06398 [cs.DC] https://arxiv.org/abs/2201.06398

[42] Zhuang Wang, Zhaozhuo Xu, Anshumali Shrivastava, and T. S. Eugene Ng. 2023. Zen: Near-Optimal Sparse Tensor Synchronization for Distributed DNN Training. arXiv:2309.13254 [cs.LG] https://arxiv.org/abs/2309.13254

[43] Yuanzhong Xu, HyoukJoong Lee, Dehao Chen, Blake Hechtman, Yanping Huang, Rahul Joshi, Maxim Krikun, Dmitry Lepikhin, Andy Ly, Marcello Maggioni, Ruoming Pang, Noam Shazeer, Shibo Wang, Tao Wang, Yonghui Wu, and Zhifeng Chen. 2021. GSPMD: General and Scalable Parallelization for ML Computation Graphs. arXiv:2105.04663 [cs.DC] https://arxiv.org/abs/2105.04663

[44] Fan Yang, Zhan Wang, Xiaoxiao Ma, Guojun Yuan, and Xuejun An. 2019. SwitchAgg: A further step towards in-network computing. In ISPA/BDCloud/SocialCom/SustainCom. https://doi.org/10.1109/ISPA-BDCloud-SustainCom-SocialCom48970.2019.00017

[45] Penghao Zhang, Heng Pan, Zhenyu Li, Penglai Cui, Ru Jia, Peng He, Zhibin Zhang, Gareth Tyson, and Gaogang Xie. 2021. NetSHa: In-Network Acceleration of LSH-Based Distributed Search. IEEE Transactions on Parallel and Distributed Systems 33, 9 (2021). https://doi.org/10.1109/TPDS.2021.3135842

[46] Lianmin Zheng, Zhuohan Li, Hao Zhang, Yonghao Zhuang, Zhifeng Chen, Yanping Huang, Yida Wang, Yuanzhong Xu, Danyang Zhuo, Joseph E. Gonzalez, and Ion Stoica. 2022. Alpa: Automating Inter- and Intra-Operator Parallelism for Distributed Deep Learning. In OSDI. https://www.usenix.org/conference/osdi22/presentation/zheng-lianmin

Table 1: Specifications of testbed.

Nodes	Item	Specification
8× Haswell	CPU	Intel Xeon E5-2630v4 10C @2.2G
	MEM	256b DDR4-2400 128GB
	NIC	NVIDIA ConnectX-5 100GbE
8× Skylake	CPU	Intel Xeon Silver 4108 8C @1.8G
	MEM	192b DDR4-2666 128GB
	NIC	NVIDIA ConnectX-5 100GbE
8× V100	CPU	Intel Xeon Silver 4112 4C @2.6G
	MEM	128b DDR4-2666 512GB
	NIC	NVIDIA ConnectX-5 100GbE
	GPU	NVIDIA V100 16GB PCIe
2× A100	CPU	AMD EPYC 7763 64C @2.5G
	MEM	512b DDR4-3200 512GB
	NIC	NVIDIA ConnectX-7 200GbE
	S-NIC	NVIDIA BlueField-2 100GbE
	GPU	NVIDIA A100 40GB PCIe

Table 2: Specifications of SmartNICs.

Model	Item	Specification
NVIDIA BlueField-2	CPU	Arm Cortex A72 8C @2.8G
	MEM	32b DDR4-3200 16GB
	NIC	NVIDIA ConnectX-6 100GbE

A Experimental Testbed

The specifications of our evaluation testbed and SmartNICs are reported in Table 1 and Table 2, respectively.

B Background

B.1 SmartNICs

In recent years, many vendors, including NVIDIA, have been actively manufacturing and promoting SmartNICs. Compared to conventional NICs, SmartNICs often provide hardware offloading or allow users to execute code on NIC cores. Based on previous research [24] and our experience, we categorize SmartNICs into the following types:

B.1.1 Fixed Function SmartNICs. They provide some hardware acceleration features, such as hardware offloading for TLS and IPSec encryption and decryption. However, these features exist in the form of ASICs within the NIC chip.

B.1.2 Programmable SmartNICs. They allow users to execute code on NIC cores. They can be further divided into on-path SmartNICs and off-path SmartNICs. (a) **On-path SmartNICs**. The NIC cores are located on the data path, and all packets are processed by these cores. These cores may already exist on conventional NICs and run RTOS, but they expose a private programming interface to users. (b) **Off-path SmartNICs**. These add additional cores to execute user code. Unlike on-path SmartNICs, these cores do not process all packets. Sometimes, these cores can be configured as a separate host, running a full-fledged Linux and programs written with DPDK [23] and RDMA Verbs [5].

Interestingly, some commodity SmartNICs may not be classified into a single category. For instance, NVIDIA's BlueField-3 [33] belongs to all three categories mentioned above. The hardware TLS and IPSec offloading part classifies it as a fixed-function SmartNIC. Its Arm Cortex A78 SoC, running Ubuntu, categorizes it as an off-path SmartNIC. Furthermore, its Data-Path Accelerator (DPA) qualifies it as an on-path SmartNIC.

In this work, we primarily discuss and optimize for on-path SmartNICs. Specifically, we utilize the Arm SoC of BlueField-2 [32] for data processing at line rate.

B.2 Advanced RDMA Features

Remote Direct Memory Access (RDMA) is a high-performance network commonly used within supercomputers and data centers. It leverages kernel bypassing, hardware offloading, zero-copy, and DMA to significantly reduce the CPU cycles consumed during high-speed transmission, thereby enhancing communication performance. We have utilized several advanced RDMA features to optimize OmNICCL.

B.2.1 Direct Cache Access (DCA). It is a non-standard feature that can be considered a hack of the PCIe controller. Standard RDMA Read/Write operations directly manipulate the memory of remote machines and generate memory transactions. However, with the rapid evolution of RDMA networks, RDMA read/write operations consume a significant amount of memory bandwidth. The latency of memory read/write operations is non-negligible for RDMA networks at the microsecond or sub-microsecond level. Therefore, some vendors have proposed and implemented DCA, such as Intel Data Direct I/O (DDIO) [15], IBM Cache Injection [8], and Arm Cache Stashing [6], which allow RDMA to directly read/write the CPU's LLC. We speculate that these implementations redirect memory access to the LLC by snooping the memory address of PCIe transactions.

B.2.2 GPUDirect RDMA (GDR). It allows direct data transfer between the NIC and GPU under the same PCIe Root Complex (RC), without the need for data to pass through the CPU and host memory, although the CPU still needs to initiate data transfers.

B.2.3 Scatter Gather List (SGL). It allows a single RDMA Send/Write Request to gather multiple data chunks located at different memory addresses into one RDMA message for transmission, or allows a single RDMA Receive Request to scatter the payload from an RDMA message to multiple different memory addresses. We found that SGL can be used in conjunction with GDR, allowing the mixing of CPU and GPU data chunks into one RDMA message.

Temporary page!

LaTeX was unable to guess the total number of pages correctly. As there was some unprocessed data that should have been added to the final page this extra page has been added to receive it.

If you rerun the document (without altering it) this surplus page will go away, because LaTeX now knows how many pages to expect for this document.