Network communication optimization of RCCL communication library in Multi-NIC systems

Shuaiming He [a], Wei Wan *[b, c], Junhong Li [b]
[a] School of computer and artificial intelligence, Zhengzhou University, Zhengzhou, Henan, 450001, China
[b] Dawning Information Industry (Beijing) Co., Ltd, Beijing, 100193, China
[c] Tsinghua University Software School, Beijing, 100084, China
[*] Corresponding author: wanwei@sugon.com

ABSTRACT

With the widespread application of deep learning frameworks, large-scale computing and GPU programming are receiving increased attention. For upper-layer applications that utilize GPUs for computational communication, such as TensorFlow and PyTorch, improving the communication efficiency of the underlying communication library is of paramount importance to enhance the overall performance of the frameworks. Among them, the RCCL (Rocm Collective Communication Library) GPU communication library, provided by the Rocm (Radeon Open Compute platform) computing platform, supports various collective communication operations and point-to-point operations. Through analysis, we have identified a problem in the initialization and usage of the ring channel network in the RCCL library, specifically in multi-network card systems. This issue results in certain network cards being unable to communicate, leading to wasted system resources. To address this problem, optimizations can be made at the code level by introducing data structures and algorithms to control the invocation of network cards. The goal is to adjust the usage strategy of multiple network cards in the ring channel network without modifying the original design concept of RCCL. After optimization, extensive evaluations were conducted using a large-scale GPU cluster. The optimized RCCL library achieved significant improvements in communication performance. Under a communication scale of 16 compute nodes and 64 GPUs, the peak bandwidth increased from 5.28GB/s to 7.78GB/s. In inter-node collective communication tests, the performance improvement reached up to 60%. The improved RCCL library provides better low-level communication performance for upper-layer applications on the Rocm computing platform, offering enhanced communication support.

Keywords: Rocm Collective Communication Library; Large scale computing; GPU programming; Collective Communication; Multi network card system.

1. INTRODUCTION

In the current field of machine learning and deep learning, large-scale model training is a focal point of attention. Large models typically have a huge number of parameters and require processing large-scale datasets to improve model performance and accuracy [1]. However, as the scale of models and datasets continues to grow, traditional single-GPU architectures are no longer sufficient to meet the demands of large-scale model training. To address the limitations of single-GPU architectures, modern computing systems commonly employ multi-GPU architectures [2]. Multi-GPU architectures distribute the computational tasks across multiple GPUs or compute nodes, leveraging the advantages of parallel computing to accelerate the training process. By fully utilizing multiple computing resources, multi-GPU architectures can significantly reduce training time and improve training efficiency.

However, despite the tremendous potential of multi-GPU architectures in large-scale model training, they face greater challenges compared to single-GPU setups. One of the challenges is the communication bottleneck between compute nodes in multi-GPU systems, where frequent exchange of parameters and gradient information is required to maintain model synchronization and consistency [3]. This communication process can become a bottleneck in the training process, limiting overall training performance improvement. Therefore, optimizing communication efficiency among multiple GPUs is an important issue with significant implications for enhancing the efficiency of large-scale model training.

In this paper, we conduct research on the RCCL release version 2.18.3 [4], a GPU communication library, analyzing the specific details of channel generation during the initialization process. We identify an issue where certain GPUs cannot be

Third International Conference on Algorithms, Microchips, and Network Applications (AMNA 2024),
edited by Joan Lu, Reggie Davidrajuh, Proc. of SPIE Vol. 13171, 131711Q
 · 0277-786X · doi: 10.1117/12.3031956

invoked in multi-GPU systems. To address this critical problem, we modify the data structures related to GPU selection in the communication channels and redefine the selection strategy for communication paths. This optimization improves bandwidth and latency performance, while minimizing wastage of system resources.

1.1 Introduction to the RCCL communication library

RCCL (Rocm Collective Communication Library) is an independent GPU standard collective communication routine library developed by the Rocm platform. It provides functionalities for data transfer and communication between multiple GPUs, aiming to support parallel computing tasks in GPU clusters and distributed computing environments [5]. It implements operations such as all-reduce, all-gather, reduce, broadcast, reduce-scatter, gather, scatter, and all-to-all communication between GPUs [6].

Through optimization by the Rocm platform, RCCL achieves high-bandwidth GPU communication on platforms using PCIe, xGMI, as well as networks using InfiniBand [7] or TCP/IP Socket. It can be used in both single-process and multi-process (e.g., MPI [8]) applications. Leading deep learning frameworks such as TensorFlow and PyTorch have integrated the RCCL communication library to leverage its efficient communication support for accelerating tasks like data aggregation in distributed machine learning tasks [9].

1.2 RCCL communication flow

RCCL plays a crucial role as an intermediate communication layer in the hardware and software stack. It provides a collection of communication API interfaces for high-level applications and deep learning frameworks, enabling control over data exchange between GPUs. The design of an efficient communication flow is a central challenge addressed by RCCL.

The communication flow of RCCL can be summarized as follows [10]:

1) Initialization: Before initiating the communication process, the NCCL communication environment must be initialized on each GPU device involved in the communication. This includes topology discovery, device connection, channel generation, and communication parameter configuration.

2) Creation of Communication Handles: Each participating process needs to create an NCCL communication handle, which serves as a management entity for communication operations.

3) Device Memory Allocation: Every process on the GPU device should allocate device memory for communication operations, which is used to store data to be sent or received.

4) Data Transfer: NCCL utilizes "communication channels" for data transfer. By assigning multiple communication tasks to different communication channels, RCCL achieves a high level of parallel data transfer.

5) Synchronization: Since data transfer is a non-blocking operation, synchronization mechanisms are employed to ensure the completion of preceding communication operations.

6) Resource Release: Upon completion of communication tasks, previously allocated device memory is released, and the communication handle and environment are destroyed.

In the aforementioned communication flow, this paper focuses on the generation and utilization of communication channels during the initialization and data transfer phases. A communication channel represents a collection of hardware components involved in data transmission, such as GPUs, NICs, and the physical links connecting them (e.g., PCIe [11]). These components are logically connected in a specific order to form a communication ring. In Chapter 2 of this paper, we analyze the search process of communication channels and highlight their limitations in multi-GPU systems.

2. RCCL TOPOLOGY DISCOVERY ANALYSIS

2.1 RCCL network topology discovery

Before conducting communication, RCCL needs to initialize the communication environment, including topology discovery and establishing communication channels. In the following example of communication between two nodes, each computing node is equipped with 4 GPUs and 1 network card, illustrating the process of RCCL establishing communication channels.

During the initialization and establishment phase of the transport layer, within each node, each process sends its own identifiers and device descriptors to other nodes, establishing an adjacency matrix based on the received information. This matrix represents the connectivity between nodes, which can be direct, indirect, or disconnected. Subsequently, each process uses this adjacency matrix to create a topological graph. The graph includes both GPU and NIC devices, which are collectively referred to as devices.

During the search process, each device performs a depth-first search starting from itself as the root. Using pre-allocated heuristic values, a set of communication paths with theoretically maximum bandwidth, referred to as communication channels, are determined. The performance of GPU-to-GPU communication is influenced by the physical location of the CPU DIE. As a result, the search produces a GPU order of 0, 1, 2, 3; 4, 5, 6, 7. Figure 2 illustrates the communication channel with the maximum bandwidth within two nodes.

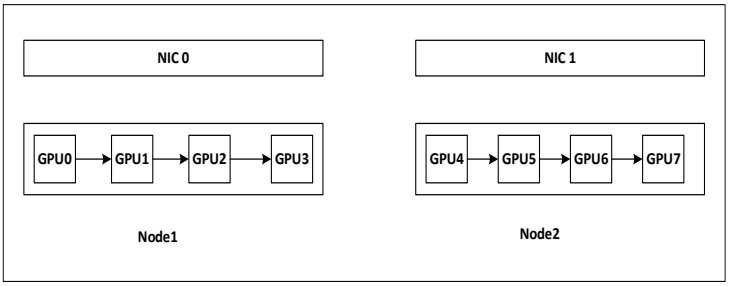

Figure 1. Single-node topology relationship diagram.

During the search process, the information about the GPU devices and NIC (Network Interface Card) enumeration within the node is stored in the data structure of the topology relationship graph (Graph). This information is saved separately as "intra" and "inter" using arrays.

After searching for communication channels within each individual node, it is necessary to connect all GPU devices in the entire computing system using the NIC. When selecting the NIC, the GPU devices located at the edge of the ring (such as GPU0, GPU3, GPU4, and GPU7 in Figure 2) will be chosen to form a path connected to the NIC devices that are connected to the same PCIe switch [12].

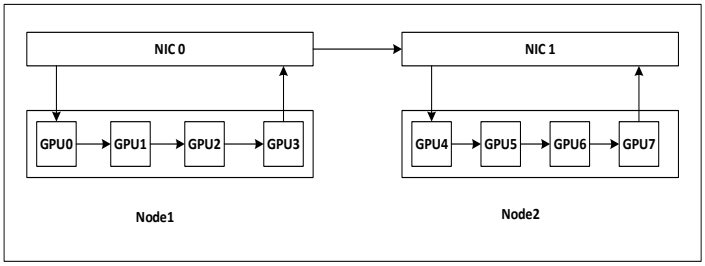

Figure 2. System topology relationship diagram.

At this point, the search for topological relationships throughout the system has been completed, and the task of establishing the transport layer is almost complete.

2.2 RCCL topology discovery for multi-NIC systems

Taking the example of each computing node having 4 GPU and 4 network card devices, when we invoke RCCL for initialization, a communication channel is also established. The difference from a single network card architecture is that in this channel, the GPU located at the edge of the node and responsible for communication with the network card will select the nearest network card device on the physical link as the tool for network communication. This is because RCCL uses a greedy strategy during the topology search to ensure that at each step, the optimal result with the maximum link

bandwidth and speed is obtained [13]. The generated channel topology structure is shown in Figure 3. Afterwards, based on the logic of channel replication, this channel will be duplicated in memory according to the user-defined quantity.

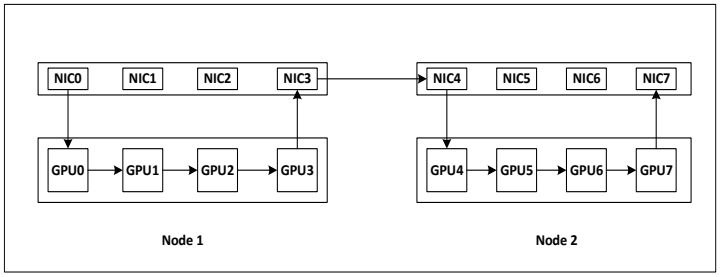

Figure 3. Multi-NIC architecture topology relationship diagram.

3. RCCL NETWORK COMMUNICATION OPTIMIZATION

Based on the analysis above, RCCL lacks optimization for multi-network card architectures. As a result, the provided example does not utilize all the network cards available, hindering highly parallelized data communication between computing nodes and significantly reducing the execution efficiency of parallel programs.

3.1 Modifications to key data structures

To address the issue of not being able to utilize multiple network cards, we have made improvements to the RCCL network card selection strategy, allowing it to utilize all the network cards within a multi-channel communication environment. Since the critical data structure, the topology relationship graph (Graph), in RCCL stores all the hardware information in the system, we can access and modify the relevant member variables in the system's topology graph to control the network card allocation during the communication channel generation process. By binding the generated communication channels with specific network cards, we can evenly distribute the network cards across each communication channel as much as possible. We also introduce an attribute called "used" to the Net member variable in the Graph, indicating the number of times it has been allocated to a communication channel. Through a quicksort algorithm, we select the network card with the least usage each time and assign it to the corresponding position in the inter array.

3.2 Optimized code

After the system generates the communication channels, we add optimized code to rewrite the inter array, aiming to optimize the invocation of multiple network cards. The optimization code is shown in Algorithm 1.

Algorithm 1*: Rewriteinter*

Input*: Graph topology system graph, number of communication channels nChannels*

Output*: ncclSuccess*

```
1.  Typedef struct {int id; int used;} Net;
2.  Net net[nnets];
3.  for i = 0 to nChannels
4.      quicksort(net[nnets].used);
5.      graph.inter[2*i] = net[0];
6.      net[0].used++;
7.      graph.inter[2*i+1] = net[1];
8.      net[1].used++;
9.  end for
```

After optimization, the internal topology structure of the system is shown in Figure 4.

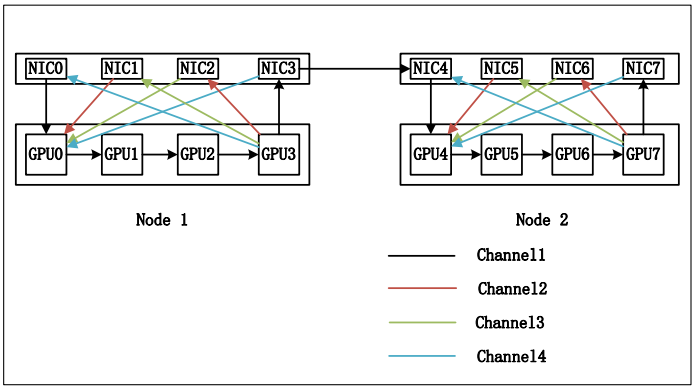

Figure 4. Optimized multi-NIC architecture topology relationship diagram.

Through the aforementioned optimization, we successfully bind each communication channel to different network cards, creating multiple ring channels to utilize multiple network cards. As a result, the input and output network cards within each communication channel are relatively independent, enhancing data transfer parallelism and improving communication efficiency.

4. EXPERIMENTAL TESTING AND VALIDATION

We conducted a series of experiments in a multi-GPU environment to evaluate the performance and effectiveness of the optimization strategy.

4.1 Experimental environment

The experiments were performed on the general-purpose nodes of Zhengzhou University's computing system. Each node was equipped with 1 processor and 4 GPU accelerator cards. Each processor consisted of 4 CPU dies, with 8 physical cores per die. The GPU accelerator cards utilized a GPU-like architecture, featuring 16GB of HBM2 device memory and multiple compute units, with a peak FP64 performance of 7.0 TFLOPS. The GPU accelerator cards were connected to the CPUs via PCIe, with a peak bandwidth of 16GB/s for data transfer between main memory and device memory. The inter-node communication was facilitated by a high-speed network connection with a bandwidth of 25GB/s.

For the GPU-to-GPU collective communication tests, we employed the rccl-tests [14] to measure and record the communication performance data. In the performance comparison data, the "befor" scenario utilized the latest release version 2.18.3 of the RCCL library, while the "after" scenario employed the RCCL library optimized according to the methods described in this paper. The test results were reported in terms of bus bandwidth, measured in GB/s. The bus bandwidth was calculated by multiplying the algorithm bandwidth with a specific collective communication operation's corresponding impact factor, as described in Equation (1).

$$bus\ bandwidth\ =\ algorithm\ bandwidth\ *\ factor \tag{1}$$

Among them, the algorithm bandwidth is the most commonly used formula for bandwidth calculation. It is calculated by multiplying the number of scanned data elements (S) with the average size of each element (E) and dividing it by the time taken (time), as shown in Equation (2).

$$algorithm\ bandwidth = \frac{S * E}{time} \tag{2}$$

The factor represents the impact factor corresponding to a specific collective communication operation. Let's take the example of the AllReduce operation to illustrate the role of the impact factor. Although the algorithm bandwidth is meaningful for measuring point-to-point communication bandwidth, it is not suitable for measuring the bandwidth of collective communication operations. This is because the theoretical peak algorithm bandwidth does not equal the hardware peak bandwidth and often depends on the number of communicating processes [15]. For an AllReduce operation involving n processes, each process needs to perform n-1 computations and n assignments. Since each step occurs on a different process except for the last input and the first output, we require 2 * (n-1) data transfers. Assuming the external bus bandwidth for each process is B, the maximum time taken for performing the AllReduce operation on S elements can be expressed as Equation (3).

$$t = \frac{S * E * 2 * (n-1)}{n * B} \tag{3}$$

By rearranging the equations, we can derive the bus bandwidth for the AllReduce operation, as shown in Equation (4).

$$bus\ bandwidth = \frac{S * E}{t} * \frac{2 * (n-1)}{n} \tag{4}$$

Further rearranging the equations, we can establish the relationship between bus bandwidth and algorithm bandwidth, as shown in Equation (5).

$$bus\ bandwidth = algorithm\ bandwidth * \frac{2 * (n-1)}{n} \tag{5}$$

4.2 Inter-Node collective communication testing

For inter-node collective communication testing, we selected the collective communication benchmarks from the rccl-tests tool. We conducted tests on different scales with node counts of 2, 4, 8, and 16. Each compute node in the test cases used 4 GPU accelerator devices, with GPU Direct RDMA option enabled. The data transfer size ranged from 1MB to 1GB, with a doubling increment for each test. The test results are presented in Table 1.

Table 1. Comparison of optimization effects before and after in different node scales (2, 4, 8, 16).

		AllReduce (GB/s)	AllGather (GB/s)	BroadCast (GB/s)	Reduce (GB/s)	Reduce-Scatter (GB/s)
2 nodes, 8 GPUs	Before	5.93	6.45	6.88	6.7	6.44
	After	8.79	9.35	10.33	9.6	9.23
4 nodes, 16 GPUs	Before	5.47	6.26	6.5	6.41	6.22
	After	8.28	8.62	9.51	9.09	8.62
8 nodes, 32 GPUs	Before	5.25	5.81	6.06	6.03	5.82
	After	7.93	8.01	8.66	8.50	7.99
16 nodes, 64 GPUs	Before	5.28	5.31	5.53	5.59	5.35
	After	7.78	7.25	7.91	7.75	7.18

The test results indicate that after applying the optimized RCCL library in inter-node communication, all five collective communication operations (AllReduce, AllGather, Broadcast, Reduce, Reduce_Scatter) in RCCL achieved significant performance improvements. Taking AllReduce as an example, which plays a crucial role in parallel computing and machine learning domains [16], it demonstrated noticeable improvements after optimization. Under the communication scale of 16 nodes, the maximum bandwidth of AllReduce increased from 6.59GB/s to 10.77GB/s, resulting in a performance improvement of nearly 63%. The optimization effect is illustrated in Figure 5.

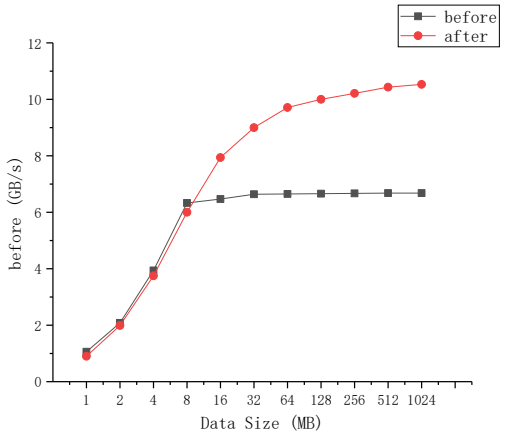

Figure 5. AllReduce optimization effect at 16-node communication scale.

4.3 Inter-Node point-to-point communication testing

For the inter-node point-to-point communication testing, we selected the "sendrecv_perf" benchmark program from rccl-tests as the reference test. We conducted tests on a scale of 16 nodes, with each node having 4 GPUs and 4 channels. However, the optimized code did not yield any improvement in point-to-point communication, as shown in Figure 6 of the test results.

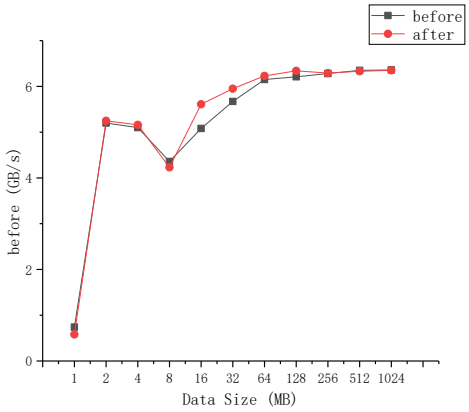

Figure 6. Sendrecv test result at 16-node communication scale.

4.4 Analysis and conclusion of test results

The test results indicate that the optimized code showed significant improvements when running collective communication functions, while it did not demonstrate any optimization effects in point-to-point communication. This is because our optimization is specifically designed for the circular channel, achieved by adjusting the key data structure "inter" that determines the selection of network cards within the circular channel. As the point-to-point communication mode does not involve the use of circular channels, the optimization effects were not observed.

5. CONCLUSION

Optimizing the RCCL library for network communication as the underlying mathematical communication library for the Rocm computing platform can enhance the overall computational performance of the platform and provide excellent low-level communication support for upper-layer application software on the Rocm computing platform. In this paper, we analyzed the process of generating ring channels during the initialization phase of the RCCL communication library, identified its shortcomings, and proposed optimizations to address the issue of certain network cards not being invoked by the default channels. By utilizing the optimized RCCL library proposed in this paper, significant performance improvements were achieved in inter-node collective communication. We have reached out to the RCCL development team, shared the idea of multi-network card optimization, and actively followed up. We hope that it can be integrated into the official code in the next release of RCCL. However, there are still many areas for further optimization in the RCCL library, such as improving the efficiency of tree-based channels, point-to-point communication, and overall performance of the RCCL library and its upper-layer applications.

ACKNOWLEDGMENT

This work was supported by the State Key R&D Program of China (No. 2021YFB0300200).

REFERENCES

[1] Lakhotia K, Petrini F, Kannan R, et al. Accelerating allreduce with in-network reduction on intel piuma [J]. IEEE Micro, 2021, 42(2): 44-52.

[2] Priscila S S, Rajest S S, Shynu T, et al. An Improvised Virtual Queue Algorithm to Manipulate the Congestion in High-Speed Network [J]. Central Asian Journal of Medical and Natural Science, 2022, 3(6): 343-360.
[3] Liu J, Huang J, Zhou Y, et al. From distributed machine learning to federated learning: A survey [J]. Knowledge and Information Systems, 2022, 64(4): 885-917.
[4] Advanced Micro Devices.GitHub-RCCL:ROCm Communication Collectives Library [DB/OL].(2023). https://github.com/ROCmSoftwarePlatform/rccl
[5] Tang Z, Shi S, Chu X, et al. Communication-efficient distributed deep learning: A comprehensive survey[J]. arXiv preprint arXiv:2003.06307, 2020.
[6] NVIDIA. Overview of NCCL[DB/OL]. (2023). https://docs.nvidia.com/deeplearning/nccl/user-guide/docs/usage/collectives.html
[7] WANG Zhi-heng. Research on InfiniBand Network Protocol Layer Software Technology [D]. Zhejiang University,2021.
[8] Hori A, Jeannot E, Bosilca G, et al. An international survey on MPI users[J]. Parallel Computing, 2021, 108: 102853.
[9] Li P, Koyuncu E, Seferoglu H. Respipe: Resilient model-distributed dnn training at edge networks[C]//ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, 2021: 3660-3664.
[10] KIDGINBROOK.CSDN-nccl[DB/OL].(2023). https://blog.csdn.net/kidgin7439/category_11998768.html
[11] A. Li et al., "Evaluating Modern GPU Interconnect: PCIe, NVLink, NV-SLI, NVSwitch and GPUDirect," in IEEE Transactions on Parallel and Distributed Systems, vol. 31, no. 1, pp. 94-110, 1 Jan. 2020, doi: 10.1109/TPDS.2019.2928289.
[12] Cubedo S A. Fast Multi-GPU communication over PCI Express[D]., 2021.
[13] Shi S, Zhou X, Song S, et al. Towards scalable distributed training of deep learning on public cloud clusters[J]. Proceedings of Machine Learning and Systems, 2021, 3: 401-412.
[14] Advanced Micro Devices. GitHub-rccl-tests [DB/OL]. (2023). https://github.com/ROCmSoftwarePlatform/rccl-tests
[15] Jian M, Alexandropoulos G C, Basar E, et al. Reconfigurable intelligent surfaces for wireless communications: Overview of hardware designs, channel models, and estimation techniques[J]. Intelligent and Converged Networks, 2022, 3(1): 1-32.
[16] Kolluru A, Shuaibi M, Palizhati A, et al. Open challenges in developing generalizable large-scale machine-learning models for catalyst discovery[J]. ACS Catalysis, 2022, 12(14): 8572-8581.