Scalable Fully Pipelined Hardware Architecture for In-Network Aggregated AllReduce Communication

Yao Liu, Junyi Zhang, Shuo Liu, Qiaoling Wang, Wangchen Dai, and Ray Chak Chung Cheung, *Member, IEEE*

***Abstract*— The Ring-AllReduce framework is currently the most popular solution to deploy industry-level distributed machine learning tasks. However, only about half of the maximum bandwidth can be achieved in the optimal condition. In recent years, several in-network aggregation frameworks have been proposed to overcome the drawback, but limited hardware information have been disclosed. In this paper, we propose a scalable fully-pipelined architecture that handles tasks like forwarding, aggregation and retransmission with no bandwidth loss. The architecture is implemented on a Xilinx Ultrascale FPGA that connects to 8 working servers with 10 Gb/s network adapters, and it is able to scale to more complicated scenarios involving more workers. Compared with Ring-AllReduce, using AllReduce-Switch improves the efficient bandwidth of AllReduce communication with a ratio of 1.75×. In image training tasks, the proposed hardware architecture helps to achieve up to 1.67× speedup to the training process. For computing-intensive models, the speedup from communication may be partially hidden by computing. In particular, for ResNet-50, AllReduce-Switch improves the training process with MPI and NCCL by 1.30× and 1.04× respectively.**

***Index Terms*— Collective communication, AllReduce, in-network aggregation, distributed machine learning.**

I. Introduction

DISTRIBUTED Deep Neural Network (DNN) becomes the common solution to perform a large-scale machine learning task, because these tasks are far beyond the computing power of a single Graphics Processing Unit (GPU) [1]. For example, it is recently reported that NVIDIA researchers have developed Megatron-LM, a Natural Language Processing (NLP) model with 8.3 billion parameters on 512 GPUs [2].

Collective communication is usually deployed to synchronize the parameters that are distributed in different working nodes [3]. In DNN, the parameter synchronization is typically an summation of gradients calculated by the Stochastic Gradient Descent (SGD) [4]. AllReduce is the most common collective communication primitive in DNN training tasks. Gradients are aggregated from working nodes, and broadcast back. With the tremendous growth of model size, the expense on AllReduce communication may even dominate the whole train process in typical tasks [5].

TABLE I
API Comparison

Library	Feature
MPI	From CPU memory to CPU memory
CUDA-aware MPI	From GPU memory to GPU memory
NCCL v1	For multiple GPUs on one machine
NCCL v2	For multiple GPUs on multiple machines

Accelerating AllReduce communication requires the comprehensive cooperation of the upper-layer application and the physical infrastructure [6]. The Application Programming Interface (API) has been evolved from classic Message Passing Interface (MPI) to the state-of-art NVIDIA Collective Communication Library (NCCL) [7], as shown in Tab. I. Efficient inter-GPU communications are achieved through the APIs that are more fit for the Distributed DNN tasks [8]. Meanwhile, the hardware is also improved to support the advanced software features [9]. NVLink and NVSwitch provide Gbps-level inter-GPU communications inside a machine [10]. Besides, NIVDIA recently acquires Mellanox aiming to speed up inter-machine communication. The Remote Direct Memory Access (RDMA) network shows a promising future in machine learning applications, due to its advantages on high performance and low latency [11].

The Ring-AllReduce framework is currently the most popular form of AllReduce communication in real applications [12]. By partially dividing the aggregation task and reorganizing the aggregation flow, Ring-AllReduce is able to achieve the optimal performance in the networks that provides identical end-to-end bandwidth communications. Nevertheless, the maximum performance of Ring-AllReduce is restricted to about half of the bandwidth, since the typical AllReduce communication requires both aggregation and broadcast. Therefore, in-node aggregation frameworks suffer the bandwidth loss.

Manuscript received March 20, 2021; revised June 10, 2021 and July 12, 2021; accepted July 13, 2021. Date of publication July 29, 2021; date of current version September 30, 2021. This article was recommended by Associate Editor M. Martina. *(Corresponding author: Shuo Liu.)*

Yao Liu and Ray Chak Chung Cheung are with the Department of Electrical Engineering, City University of Hong Kong, Hong Kong (e-mail: liu.yao@cityu.edu.hk; r.cheung@cityu.edu.hk).

Junyi Zhang, Shuo Liu, and Qiaoling Wang are with Huawei Company, Shenzhen 518129, China (e-mail: zhangjunyi5@huawei.com; liushuo15@huawei.com; wangqiaoling@huawei.com).

Wangchen Dai is with ByteDance Ltd., Shenzhen 518000, China (e-mail: w.dai@my.cityu.edu.hk).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TCSI.2021.3098841.

Digital Object Identifier 10.1109/TCSI.2021.3098841

In contrast to the in-node aggregation frameworks that typically aggregate the parameters inside a Parameter Server (PS) or a worker, several in-network aggregation frameworks that aggregate the data in the network device have been proposed recently in order to better utilize the bandwidth. Mellanox's SHARP^{TM} provides the fine control of all the data flows, but it requires the user to deploy the whole set of program interface, communication protocol, and a quantum switch [13]. The iSwitch [5], SwitchML [14] and NetReduce [15] frameworks extends the standard Ethernet network with the in-network aggregation feature on a Field Programmable Gate Array (FPGA) or a programmable switch.

In previous works, hardware architecture information has seldom been introduced, despite the crucial role of hardware switch. According to the limited information disclosed in previous works, most of the architectures have drawbacks. The architecture of iSwitch cannot avoid the bandwidth loss; SwitchML cannot be fully customized due to the dependence on the programmable switch chip; NetReduce suffers the scalability problem that requires both the commodity network switch and the FPGA. We propose and implement a hardware architecture called AllReduce-Switch based on the NetReduce framework that provides a comprehensive setup for industry-level applications. Note that this idea of AllReduce-Switch is not limited to the NetReduce framework, and it is compatible with other Ethernet-based frameworks with necessary modifications. The features of our work are summarized as follows:

- *Forwarding, aggregation and retransmission are supported.* AllReduce-Switch identifies the aggregation frame, aggregates the payload, and reassembles the aggregated frame with frame header and payload. Frames not for aggregations are directly forwarded to the target port according to the destination Media Access Control (MAC) address. Additionally, retransmission is supported by buffering necessary aggregation information to tackle realistic exceptions. Up to 512 aggregated frames and the related input frames can be buffered. A cyclic maintenance mechanism of the buffer is introduced in this paper.
- *Fully-pipelined architecture guarantees a deterministic aggregation latency and no bandwidth loss.* The complete processing flow of AllReduce-Switch is divided into 4 stages, which are separated by First-In-First-Outs (FIFOs). Each stage independently operates to maintain the maximum throughput. According to the input frame type, the data flow may skip some parts of the stages. The 64-bit data flow is processed with 200 MHz, to satisfy the physical requirement from the 10 Gb/s SPF+ Physical Layer (PHY) ports.
- *High scalability.* AllReduce-Switch supports online configuration for the aggregation port number, ranging from 2 to 8. Each port can be configured as an aggregation port or not. Fixed-point and floating-point additions are both supported for aggregation. The idea of AllReduce-Switch can be also extended to support more ports.

The rest of this paper is organized as follows. In Section II, we briefly introduce existing frameworks for AllReduce communication. In Section III, we introduce the design specification and challenges of AllReduce-Switch. In Section IV, we introduce the detailed implementation of the hardware architecture on an Ultrascale FPGA device. In Section V, we analyze the hardware performance and the acceleration rate in real applications. Finally, in Section VI, we do the conclusion.

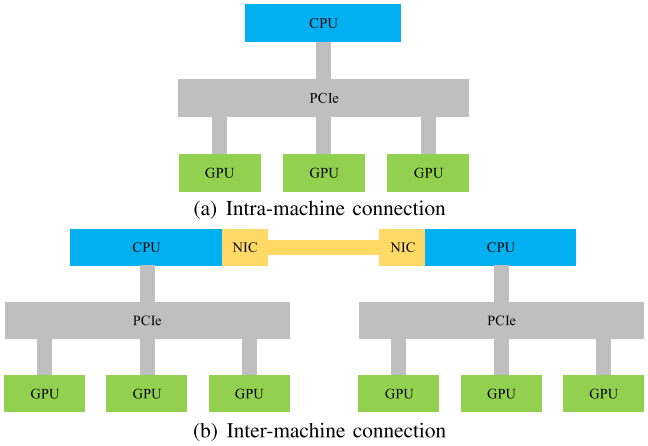


Fig. 1. Physical connections of machine learning clusters.

II. Background

In general, the physical infrastructure can be grouped into single-machine multiple-GPU, multiple-machine single-GPU and multiple-machine multiple-GPU scenarios. The topology of AllReduce communication is built on top of the physical infrastructure, and the physical connection restricts the overall performance. The propagation delay and the efficient bandwidth are usually used to evaluate the performance. The efficient bandwidth in typical node is defined by dividing the AllReduce data size with the time consumption.

A. Physical Connection

Fig. 1 shows the ordinary intra-machine and inter-machine connections. The Central Processing Unit (CPU) coordinates the operations of the components, but is not responsible for the machine learning task. Consequently, the development trend is to bypass the CPU, and to enhance the direct connection between GPUs. For the intra-machine connection, the development of Peripheral Component Interconnect Express (PCIe) interconnection significantly facilitates the direct connection between GPUs [16]. For the inter-machine connection, network communication via Network Interface Controllers (NICs) is usually employed [17]. The innovative protocol such as RDMA shows the overwhelming efficiency over the traditional TCP/IP protocol by avoiding the unnecessary caching [18].

B. Tree Topology

The physical connection is naturally a tree topology, which is an plausible choice for AllReduce communication, as shown in Fig. 2. In tree topology, messages are aggregated in the root node, and the internal node relays the message from its child node.

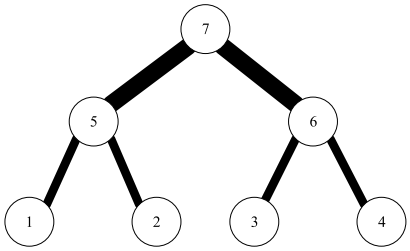


Fig. 2. Tree topology of communication.

Tree topology is optimal only if the bandwidth of the internal node is larger than the sum of bandwidth of its child nodes, which is known as the fat-tree topology [19]. That is because the full-speed message transmission from the child nodes leads to the message congestion in their parent node. In this case, for a tree with n nodes, the root node needs to have the sufficient bandwidth to process messages from the rest $(n-1)$ nodes, which is identical to the star topology. The bandwidth pressure of the root node can be alleviated if the internal node partially aggregates the messages from its child nodes. However, the bandwidth of the internal node still needs to be larger than the sum of bandwidth of its child nodes.

Assume that a tree topology with n nodes is constructed by k layers, and each node aggregates the messages from its m child nodes, and it has $k \approx \log_m n$. Assume that the delay of message processing and transmission in a node is τ, and B is the bandwidth of the communication channel between nodes. The propagation delay of AllReduce communication in this tree is $2\log_m n\tau$, and the bandwidth is $\frac{B}{2}$, because the All-Reduce communication consists of aggregation and broadcast.

C. Ring Topology

Ring-AllReduce uses ring topology to cyclically transmit messages. The working node unidirectionally sends messages to its successive node. The aggregation task is sliced and distributed into all the working nodes. As shown in Fig. 3, for parameter set A, Node 1 begins to send A1, and Node 2 sends the partial aggregated parameter with A1+A2. After 3 times of transmission, Node 4 finally calculates the aggregated value, and sends it to Node 1. After 3 times of propagation, the aggregated value is received by all the nodes. Meanwhile, the parameter aggregation flow can be adjusted to process different parameter sets in parallel as well. In the example of 4 nodes in Fig. 3, 4 parameter sets are in the aggregation process at the same time, and the propagation times for each parameter set is 6. The parallel nature of DNN is highly compatible with the ring topology, and therefore Ring-AllReduce is the most popular framework in practice [20].

Assume all the nodes are identical, and the delay of message processing and transmission in a node is τ. For a topology with n nodes, the delay of aggregation and broadcast is $2(n-1)\tau$. n^2 messages are aggregated into n messages in parallel. Notate B as the bandwidth of the communication channel between nodes, so that the efficient bandwidth of Ring-AllReduce communication is $\frac{nB}{2(n-1)}$.

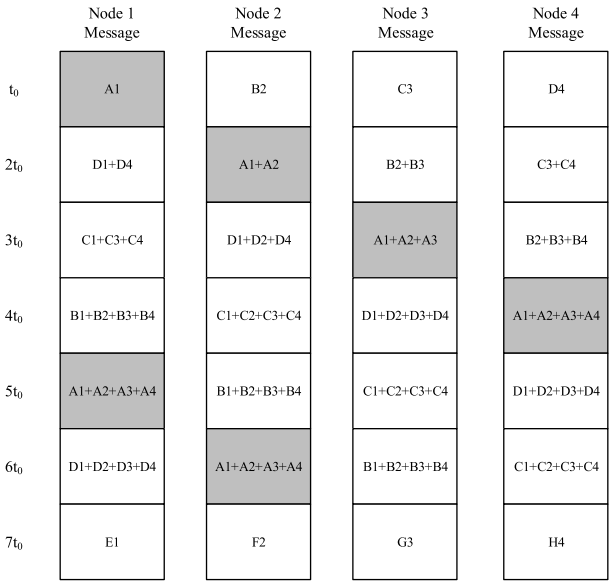
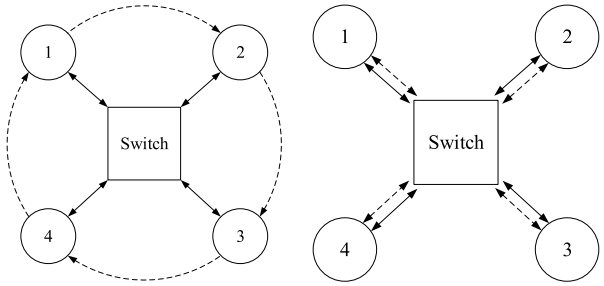


Fig. 3. Message transmission in Ring-AllReduce.



(a) Ring-AllReduce with a network switch (b) In-network aggregation

Fig. 4. Physical connection and topology of a network switch. The solid arrow shows the physical data flow, and the dotted arrow shows the communication topology.

In the case of non-identical nodes, although the propagation delay of the communication is related to all the channels within the ring, the slowest node restricts the efficient bandwidth of communication. The efficient bandwidth of Ring-AllReduce communication is $\frac{nB_{min}}{2(n-1)}$. The communication flow stalls for the response of the slowest node. In this case, hierarchical topology with multiple rings that is considered with practical conditions is a better choice [21], [22].

In real applications, the working nodes in Ring-AllReduce are not necessarily wired like a ring. Instead, the fully-connected network that provides flexible end-to-end connections is preferred [23]. A network switch is typically employed to support the inter-machine connection [24], as shown in Fig. 4(a), although the switch is not regarded as a communication node in the topology.

D. In-Network Aggregation

In the frameworks that use in-network aggregation, the communication topology coincides with the physical connection, as shown in Fig. 4(b). The aggregation is typically processed

TABLE II
TOPOLOGY COMPARISON

	Propagation delay	Efficient bandwidth	Bandwidth req. in critical node
Tree topology	$2\log_m n\tau$	$\frac{B}{2}$	mB
Ring topology	$2(n-1)\tau$	$\frac{nB}{2(n-1)}$	B
In-network agg.	τ	B	nB

* B represents the node-to-node communicate bandwidth. τ represents the node-to-node propagation delay. n represents the node number. m represents the child-node number per node.

in the network switch, so that each node only sends and receives one message to do the aggregation once. As long as the switch promptly aggregates the messages, the whole AllReduce communication works with the full bandwidth. In addition, the propagation delay and the efficient bandwidth of the in-network aggregation are independent from the number of nodes or child-nodes.

Tab. II lists the comparison of tree topology, ring topology and in-network aggregation. The propagation delay of tree topology is better than that of ring topology, and the efficient bandwidth is slightly smaller. However, tree topology requires a complex offloading mechanism between PS and workers, which is not as flexible as frameworks using peer-to-peer communication [17]. Consequently, ring topology, *i.e.* Ring-AllReduce, with the desirable scalability and the similar performance is more preferable in real applications. Nevertheless, only about half of the bandwidth can be used. Unlike tree and ring topologies, frameworks of in-network aggregation fully utilizes the bandwidth. This brings the design challenges for the network switch, since it is the only critical node of the whole framework.

E. Existing Frameworks

Currently, three frameworks using in-network aggregation based on Ethernet have been proposed. At the beginning, Li *et al.* proposed iSwitch based on User Datagram Protocol (UDP) connection [5]. Since UDP does not aim for high-performance applications, iSwitch suffers several drawbacks, such as: a) CPU is involved in the data offloading; b) complex mechanisms like congestion control are not supported. Therefore, UDP is not the optimal choice for applications that requires for huge data transmission, such as large-scale distributed machine learning tasks. iSwitch is implemented on a NetFPGA board with 4 × 10 Gbit/s SPF+ ports. The hardware architecture is not fully optimized either, so that the platform may suffer the bandwidth loss when the task is performed at the full bandwidth.

Sapio *et al.* proposed SwitchML based on UDP using a programmable switch that supports 10 Gbit/s and 100 Gbit/s connections [14]. SwitchML extends the UDP protocol with the retransmission feature that needs the data caching from the worker, which introduces extra latency.

Liu *et al.* proposed NetReduce base on RDMA over Converged Ethernet (RoCE) [15]. NetReduce specifies a customized NCCL header that lies between Transport Layer and Session Layer. As the layer-4.5 header, the NCCL header

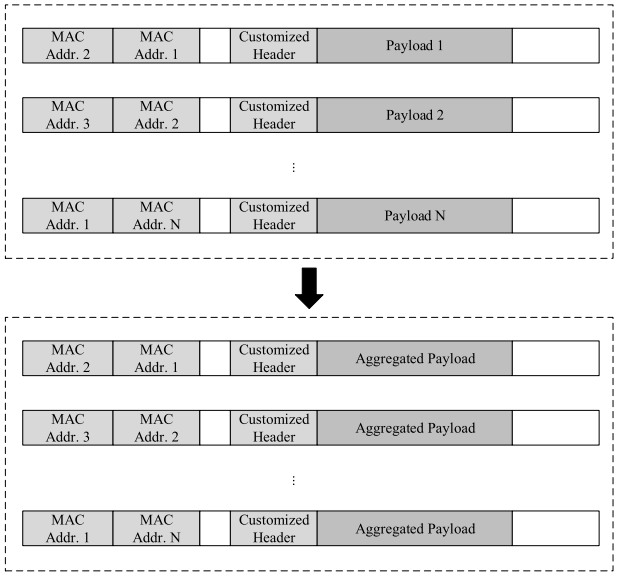


Fig. 5. In-network aggregation in NetReduce.

TABLE III
IN-NETWORK AGGREGATION FRAMEWORK COMPARISON

	Protocol	Platform	Retransmission
iSwitch [5]	UDP	FPGA	Not supported
SwitchML [14]	UDP	Programmable switch	Buffer in workers
NetReduce [15]	RoCE	Switch and FPGA	Buffer in FPGA
	Port No.	**Comm. bandwidth**	**Bandwidth loss**
iSwitch [5]	4	10 Gbit/s	Yes
SwitchML [14]	Not mentioned	10 Gbit/s, 100 Gbit/s	Not mentioned
NetReduce [15]	6	100 Gbit/s	No

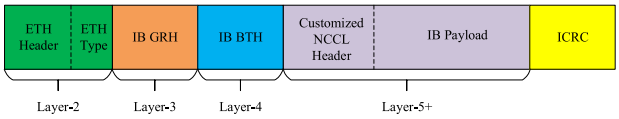


Fig. 6. Frame format of NetReduce over RoCE v1. GRH is the header of Network Layer, and BTH is the header of Transport Layer in RoCE v1. NetReduce utilizes QP, PSN and OpCode fields in GRH and BTH, together with the customized NCCL header to facilitate the distributed training process.

provides necessary parameters for NCCL applications, such as message ID, *etc*. With the NCCL header and the inherent fields in the RDMA header, (*e.g.* Queue Pair (QP), Packet Sequence Number (PSN)), aggregation frames are identified from normal frames, as shown in Fig. 5. The hardware drawback of NetReduce is that the forwarding and aggregation flows are separately processed in a network switch and an FPGA, which restricts the scalability. The comparison of the three frameworks is listed in Tab. III.

III. ALLREDUCE-SWITCH DESIGN

We propose a scalable hardware architecture that is compatible with NetReduce over the RoCE v1 protocol [25]. The brief frame format is shown in Fig. 6.

A. Specification

Tab. IV briefly lists the specification of AllReduce-Switch. First of all, all the frames follow the Ethernet IEEE

TABLE IV
SPECIFICATION OF ALLREDUCE-SWITCH

Parameter	Information
Platform	Xilinx Ultrascale XCVU190-FLGA2577-2-E
PHY port	8 × 10 Gbit/s SPF+ ports
Function	Forwarding, aggregation, retransmission
Protocol for aggregation frames	RoCE v1
Aggregation payload size	Write First: 20-byte header, 1004-byte payload Other OpCodes: 1024-byte payload
Aggregation operation	32-bit Fixed-point addition, 32-bit floating-point addition
Aggregation buffer size	4 frame flows × 128 frames = 512 frames

802.3 standard. The switch not only aggregates the aggregation frames, but also forwards other types of frames according to the destination MAC address. The length of the customized NCCL header is set as 20 bytes to provide necessary programming parameters for the upper-layer NCCL programs. In the IEEE 802.3 standard, the Maximum Transmission Unit (MTU) cannot exceed 1500 bytes. For the convenience of upper-layer use, we set a fixed length of 1024 bytes for the layer-5 payload in the aggregation frame, including the NCCL header and the aggregation payload, despite the fact that the RoCE v1 protocol supports variable payload length.

Second, AllReduce-Switch and upper-layer applications fully utilize the advantages brought by RDMA. The synchronization of the huge number of parameters in a distributed training process requires for several frames to transmit the data, and RDMA provides efficient mechanisms like OpCode to control the communication data flow. In AllReduce-Switch, the in-network aggregation supports OpCodes of "Write First", "Write Middle", "Write Last" and "Write Last with Immediate". A typical aggregation flow always begins with a frame of "Write First", and ends with a frame of "Write Last" and "Write Last with Immediate". The rest frames in the flow belong to "Write Middle". The RDMA NIC manages the flow according to the header fields, especially QP and PSN. In the same flow, QP maintains the same; the 24-bit PSN value is cyclically increased by 1 when the NIC sends a new frame. Note that the PSN of a flow does not necessarily begin with 0, but it is manipulated by the NIC. Consequently, the NCCL header is not compulsory in frames that are not "Write First" frames. Instead, the aggregation frames can be identified by QP and PSN. The capacity ratio for payload in the aggregation frames are hence improved, which is 90.5%, 93.6%, 93.6% and 93.3% in "Write First", "Write Middle", "Write Last" and "Write Last with Immediate" frames respectively. Note that each NIC connecting to AllReduce-Switch independently manages the flow with independent QP and PSN values.

The aggregation flow is the operation of all the workers. In principle, the upper-layer application should guarantee the smooth operation of AllReduce-Switch. However, robust mechanism such as data buffering and retransmission should be supported to tackle realistic problems. For example, NICs send aggregation frames that belong to the same aggregation identification with significant timing intervals; the aggregation

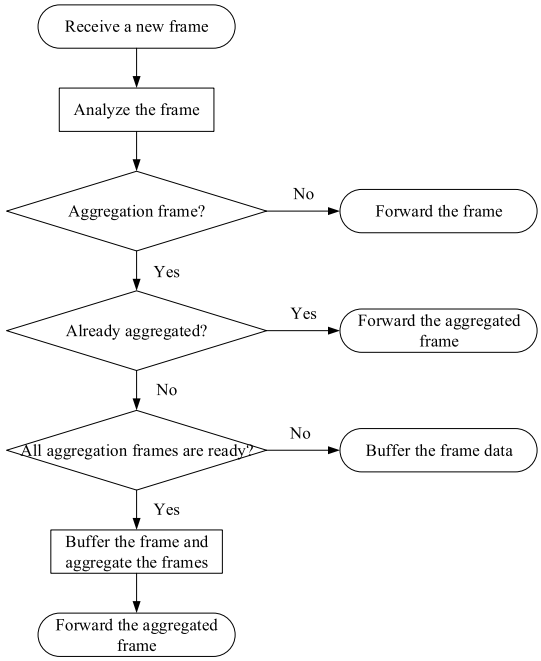


Fig. 7. Process flowchart of AllReduce-Switch.

frames are congested by forwarding frames; exceptional cases such as the package loss may occasionally occur. Redoing a aggregation operation is costly, which requires the cooperation of all the workers. Instead, the worker only needs to send the previous aggregation frame; the switch identifies the frame, and retransmits the buffered aggregated frame to the typical worker. Ideally, it is better if AllReduce-Switch is able to buffer as more frames as possible. Note that for one aggregation process, 9 frames including 1 aggregated frame and 8 aggregation frames need to be buffered, so that at least 9 kB on-chip memory needs to be consumed to buffer one time of aggregation. In NetReduce, the default setting is to buffer 4 aggregation flows with 128 frames each. Our target FPGA platform Xilinx Ultrascale XCVU190-FLGA2577-2-E can tolerate this buffer size, so that we adopt the setting in NetReduce.

The processing flow of AllReduce-Switch is shown in Fig. 7. Note that the switch processes the frames from all the 10-Gigabit Ethernet (10GE) ports in parallel. The aggregation flow starts only if all the aggregation frames with the same index are buffered. The same index here means that: the switch judges that the input frames from all the ports are of the same order in the same flow. The details of the processing method is introduced in Section IV.

B. Design Challenges

The major design challenge is how to maintain the maximum bandwidth while supporting all the features in the specification. Pipeline is an desirable architecture to promptly process the frames. Apparently, the processing throughput must be higher than the sum of the bandwidth of all the data flows, so that timing convergence becomes the most crucial problem in the implementation.

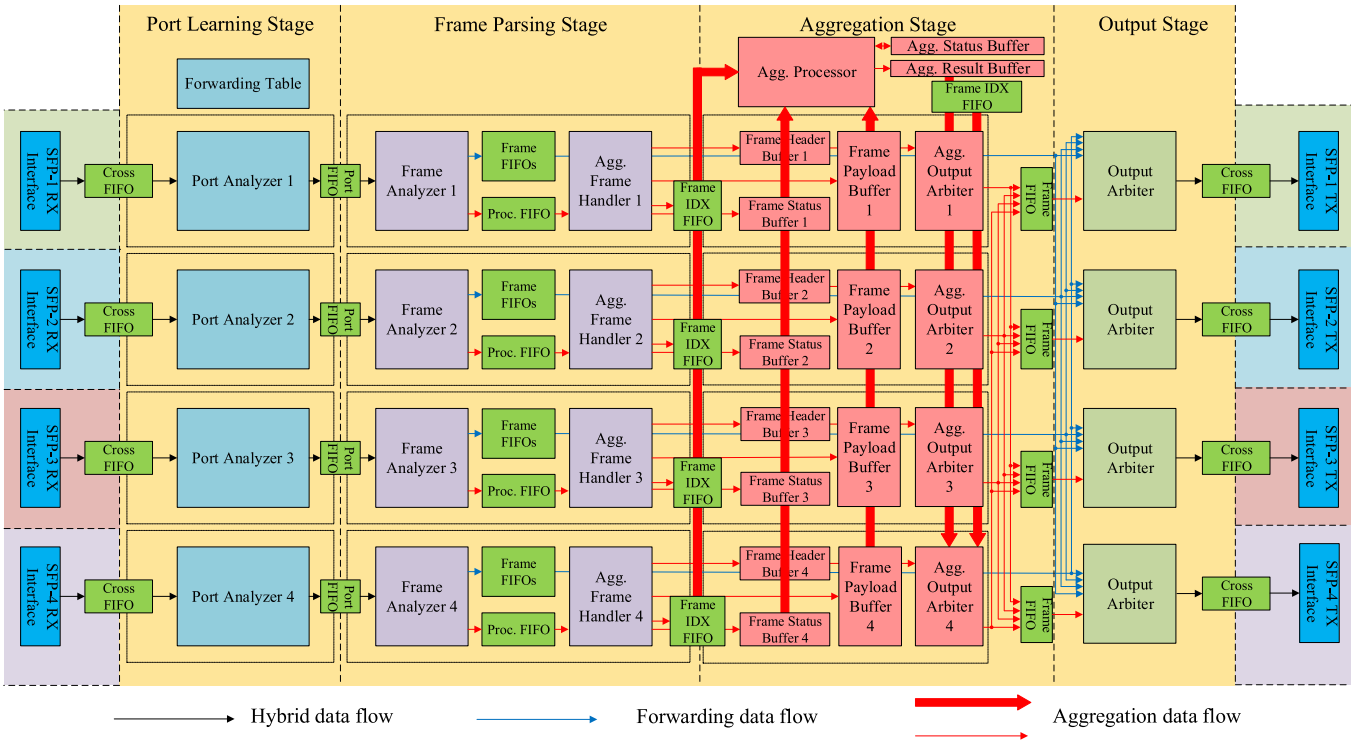


Fig. 8. Architecture of AllReduce-Switch with the data flow.

NetReduce raises harsh requirements on the memory size and speed. Since one aggregation frame size is over 1 kB, each port needs at least 512 kB memory that operates with independent interfaces. Although FPGA is advantageous in abundant on-chip memory resources, the memory usage will still be the bottleneck when the port number increases. Additionally, the fixed location of Block Random Access Memory (BRAM) restricts Place and Route (P&R) performance. When the memory utilization increases, the congestion is unavoidable. In this case, some critical paths with long delay may appear, which significantly reduces the maximum operation frequency.

The hardware should provide sufficient bandwidth for every kind of data flow in every end-to-end connection. Mechanisms, such as data buffering, frame assembling, *etc.*, will increase the P&R difficulty when the architecture scales up. Consequently, the cross Super Logic Region (SLR) design is inevitable on an Ultracale FPGA device. Since the interconnection between SLRs is much slower than the route inside an SLR, the Electronic Design Automation (EDA) tool is likely to have poor performance in the automatic implementation. In practice, the cross-SLR design usually involves systematic efforts, such as adding cross-SLR logics, manual floor-plan, specific constraints on cross-SLR modules, and so on.

IV. Hardware Architecture

The architecture of the proposed switch is shown in Fig. 8. The major modules of the switch are divided by functions in the data flow. The blue arrow shows the forwarding data flow, and the red arrow shows the aggregation data flow. The black arrow indicates the represented data flow is mixed with aggregation data and forwarding data. The broad red arrow shows the gathering or the scattering flow of the aggregation data. Due to the space limit, Fig. 8 illustrates the architecture by using the case of 4 ports.

TABLE V
Features of FIFOs

FIFO modules	CLK	Content
Cross FIFO	Cross freq.	Frame data.
Port FIFO	Processing	Frame data, dest. port.
Proc. FIFO	Processing	Frame data, dest. port, QP, PSN, OpCode.
Frame IDX FIFO	Processing	Frame index, OpCode.
Frame FIFO	Processing	Frame data.

From the receiving (RX) port to the transmission (TX) port, a frame is processed by 4 stages: Port Learning Stage, Frame Parsing Stage, Aggregation Stage and Output Stage. Different stages are separated by FIFOs, so that the data is processed in the way of pipeline. Tab. V illustrates the different FIFO modules and their features. FIFOs plays a crucial role in the data flow. Their functions include cross-clock-domain synchronization, stage pipeline, temporary data storage, *etc*.

A. 10G Ethernet Interface

Xilinx Vivado tool suite provides the 10GE Intellectual Property (IP), which is able to perform the conversion between the parallel Advanced eXtensible Interface (AXI) data stream and the high-speed serial differential signal from the SPF+ port. In Fig. 8, the 4 10GE ports are driven by 4 independent clocks. Practically, some transceivers may also share the same clock, which needs the proper PCB design and pin assignment.

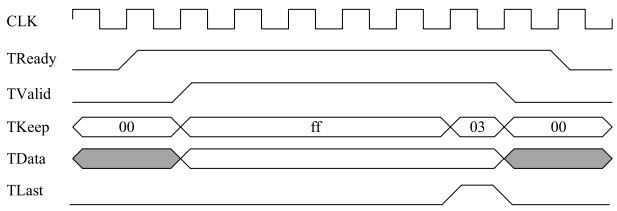


Fig. 9. AXI data stream.

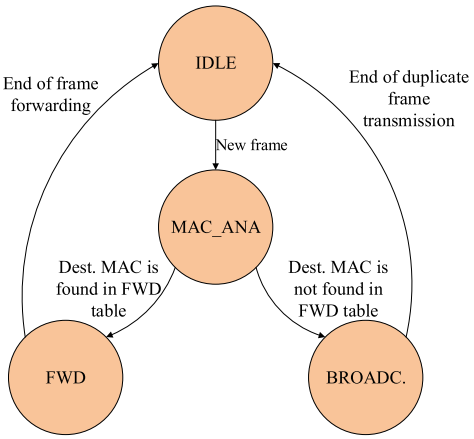


Fig. 10. FSM of the Port Analyzer.

The AXI data stream used in 10GE IP is little-endian byte order with the width of 64 bits. The clock frequency of transceivers is 156.25 MHz to achieve 10 Gbit/s bandwidth, which demands the processing clock frequency to be larger than that. In AllReduce-Switch, the clock frequency is 200 MHz.

The data flow within the AllReduce-Switch follows the AXI data format, which is shown in Fig. 9. TReady is high-active, and indicates the validity of data. The empty signal of FIFOs can be used to generate TReady. TData is 64-bit wide, and it is always filled from the lowest significant bytes. TKeep is 8-bit wide, and indicates the valid bytes of TData. In Fig. 9, TKeep is 8'h03 in the last clock cycle, which means the lower 2 bytes are valid in TData. TLast indicates the last clock cycle of a data flow. In one frame, TLast is triggered only once.

B. Port Learning Stage

The major task in Port Learning Stage is to identify the destination port of the frame. A forwarding table that stores the MAC information of the ports is globally maintained by all the Port Analyzers.

The brief Finite State Machine (FSM) of the Port Analyzer is shown in Fig. 10. In the *MAC_ANA* state, the ETH header of the frame is analyzed. The source MAC field is used to update the forwarding table, and the destination MAC field is used to identify the target port. If the destination MAC matches the value in the forwarding table, the whole frame together with the destination port information will be forwarded to the following FIFO. When the destination MAC does not match any value in the forwarding table, the broadcast will be triggered. The Port Analyzer will duplicate the frame for the times of the port number, and label the frame with the corresponding port index, for the further processing in the further stages. The process is similar to a learning process, so it is called Port Learning Stage.

The port learning process should be completed with forwarding frames, because duplicate aggregation frames will cause chaos in the switch. In practical scenarios, the NIC automatically sends out frames. The port learning process can be quickly completed after a new port is inserted, without the assistance from the upper-layer program.

C. Frame Parsing Stage

The tasks in Frame Parsing Stage include parsing the frame header, splitting the data flow, judging the aggregation information and maintaining the aggregation frame status. Two subordinate stages can be divided, which are mainly controlled by two modules respectively. The Frame Analyzer undertakes the first half, processing all the frames. The Aggregation Frame Handler is in charge of the second half, further processing the aggregation frames.

1) Frame Analyzer: The Frame Analyzer parses the frame header and splits the data flow. First, it judges whether an input frame is an aggregation frame. If true, QP, PSN and OpCode are obtained from the related fields. After that, the data is piped to the aggregation data flow or the forwarding data flow, according to the frame type. The data flow and the schematic of the Frame Analyzer is shown in Fig. 11.

There are 6 possible scenarios when DEC_FSM analyzes the frame header. The corresponding operations regarding the conditions are listed in Tab. VI. Note that QP and PSN from a typical worker are unrelated with other workers. Each Frame Analyzer independently obtains and utilizes QP and PSN.

Due to the complexity of the frame header, a variable number of clock cycles are needed for DEC_FSM to parse the frame header. Meanwhile, the data flow cannot be suspended. Consequently, DEC_FSM reads out the data from the previous FIFO, and temporally saves the frame in the following FIFO. After a whole frame is read out from the previous FIFO, DEC_FSM saves the related frame header information in the Decoder Information FIFO. PROC_FSM is then activated, and DEC_FSM prepares for the next frame.

During the *IDLE* state, PROC_FSM checks the status of Decoder Information FIFO. If data exists, it reads out the frame header information. In the *JUDGE* state, the data flow is judged. If it is a forwarding flow, the frame data is piped to Output Stage; if not, the frame data together with the frame header information are passed to the Aggregation Frame Handler.

2) Aggregation Frame Handler: The Aggregation Frame Handler deals with aggregation frames. By looking up the Frame Status Buffer, it decides whether the aggregation package was processed previously. If so, a retransmission will be triggered; if not, it buffers the aggregation frame data and updates the Frame Status Buffer.

The brief FSM of the Aggregation Frame Handler is shown in Fig. 12. In the JUDGE stage, the Aggregation Frame Handler calculates the index address by analyzing PSN, and

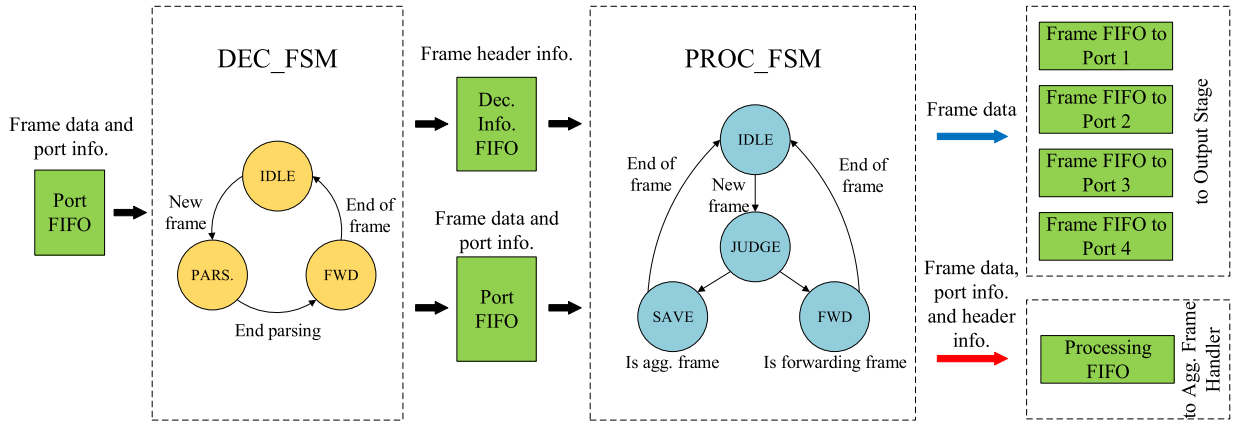


Fig. 11. Data flow in the Frame Analyzer.

TABLE VI
OPERATIONS IN FRAME HEADER PARSING

Frame type	OpCode	Aggregation field in NCCL header	QP	Operation
Not RoCE v1	-	-	-	Forward
RoCE v1	Not supported in aggregation	-	-	Forward
RoCE v1	Write First	Match	-	Aggregation flow; Record QP as the aggregation QP; Record PSN and OpCode.
RoCE v1	Write First	Not match	-	Forward
RoCE v1	Wrist Middle / Write Last / Write Last With Immediate	-	Match the aggregation QP	Aggregation flow; Record PSN and OpCode.
RoCE v1	Wrist Middle / Write Last / Write Last With Immediate	-	Not match the aggregation QP	Forward

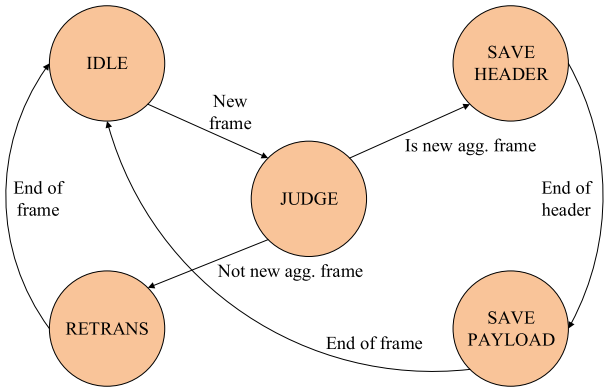


Fig. 12. FSM of the Aggregation Frame Handler.

then looks up the Frame Status Buffer, to judge whether the aggregation frame is new or not. If the frame with the index is aggregated before, a retransmission flow occurs; after the current frame is read out from the previous FIFO, the Aggregation Frame Handler sends the index of the frame and OpCode to the Frame Index FIFO. If the frame is new, the Aggregation Frame Handler determines the respective length of frame header and the payload according to OpCode, and buffers the frame data in the Frame Header Buffer and the Frame Payload Buffer respectively; after that, the value corresponding to the index in the Frame Status Buffer is set to 1.

The Frame Status Buffer is maintained by several floating pointers. Since only "Wirte First" frames contains the customized NCCL header, PSN is the only parameter that determines the address. The cyclic update mechanism of floating pointers is shown in Fig. 13.

After initialization (reset or power-on), the 4 pointers *p0*, *p1*, *p2* and *p3* point to the base address that is 0, 128, 256 and 384 respectively. The point *pnext* always points to *p0*. An aggregation frame of "Write First" indicates the start of a new aggregation frame flow, and current index is the base PSN pointed by pnext. After that, all the pointers cyclically move to the next base address. When the fifth aggregation frame flow begins, the column addressed from 0-127 needs to be refreshed.

When receiving other types of aggregation frames, the address needs to be calculated by PSN. Due to the intrinsic property of the NIC, frames that belong to the same QP increase PSN by 1 in a frame flow. The value of PSN is cyclically changed from 0 to 24'hffffff. In the *JUDGE* state, PSN is successively compared with the base PSN pointed by *p3*, *p2*, *p1*, *p0*. If the absolute distance is less than 128, the base address is determined, and the distance is the offset address.

Note that the distance may be a minus value, since PSN cyclically increases. In implementation, the practical process is more complicate. Complement a highest bit to the base PSN, and all the base PSN values are initialized with 25'h1ffffff, which is larger than any valid PSN. First, PSN is compared with base PSNs, and the base address is determined. Note that the strict comparison order from *p3* to *p0* is crucial. If PSN is smaller than all the base PSNs, it belongs to the column pointed by *p3*. After that, the absolute distance is calculated

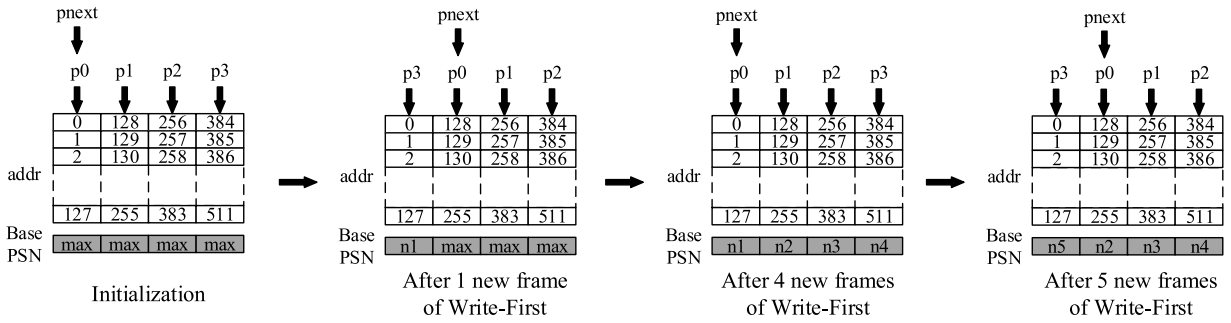


Fig. 13. Cyclic update mechanism of floating pointers for the frame status buffer.

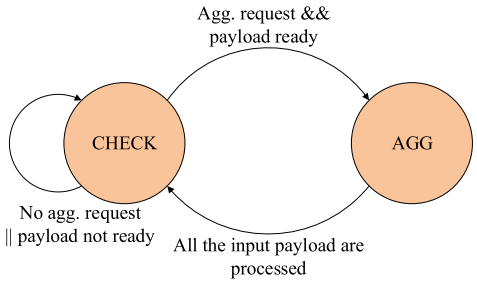


Fig. 14. FSM of the Aggregation Processor.

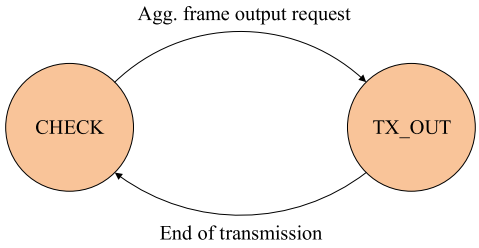


Fig. 15. FSM of the Aggregation Output Arbiter.

by subtracting the base PSN from {1'h1, PSN}. The '1' at the complement bit avoids the underflow. After the subtraction, the highest bit is chopped off. For example, the current base PSNs are 25'h0ffffe0, 25'h0fffff0, 25'h0ffffff and 25'h1ffffff, and PSN of the current frame is 24'h0. *pnext* points to 384. Then the index of the frame is 257.

Because of the independence nature of NICs, different Aggregation Frame Handlers independently maintain their own Frame Status Buffers. Therefore only the frame index is the valid parameter for the following stages to process.

The maintenance of the Frame Status Buffer is crucial in the switch. When a new flow begins with a "Write First" frame, the column pointed by *pnext* should be cleared, which requires 128 clock cycles. Since an aggregation frame is always larger than 1024 bytes, the processing time is always larger than 128 clock cycles. The clear procedure can be done in parallel with the processing without increasing extra latency.

D. Aggregation Stage

Aggregation Stage can be further divided into two subordinate stages, aggregation and output. An aggregation flow needs both stages, but an retransmission flow only have the second one.

Note that the process in Aggregation Stage is essentially a combination and redistribution with all the branch flows. Therefore the 10 Gbit/s bandwidth can still be maintained.

1) Aggregation Processor: The Aggregation Processor deals with the aggregation process. The brief FSM of the Aggregation Processor is shown in Fig. 14. During the *CHECK* state, the Aggregation Processor inquires the status of the Frame Index FIFOs with the Round-Robin mechanism. When data exists in a branch, the Aggregation Processor checks the Aggregation Status Buffer by the frame index, and increases the value by 1. When the value exceeds the port number for AllReduce communication, the Aggregation Processor aggregates the data and clear the status value in the Aggregation Status Buffer. If the value is still lower than the threshold, it means that the aggregation data is not ready; the Aggregation Processor updates the value in the Aggregation Frame Buffer, and inquires the next branch.

2) Aggregation Output Arbiter: The FSM of the Aggregation Output Arbiter is shown in Fig. 15. In the *CHECK* state, it checks the retransmission flow from the Aggregation Frame Handlers and the aggregation flow from the Aggregation Processor. If the request exists, FSM goes to the *TX_OUT* state. For the retransmission, the aggregated frame is sent to the target branch according to the destination port information. For the aggregation, the aggregated frame is broadcast to all the branches. The inquiry uses Weighted Round-Robin mechanism. The weight of the aggregation flow is 1/2, and the retransmission requests from the branches equally shares the rest 1/2.

The major process in the *TX_OUT* state is packaging the frame with data from the Frame Header Buffer and the Aggregation Result Buffer, and calculating the Invariant Cyclic Redundancy Check (ICRC) value at the same time. The calculation of ICRC follows the RoCE v1 standard, which is listed as follows:

- Remove the ETH header.
- Add 64 bits of all "1" before the GRH header, to replace the non-existing LRH header.
- In the GRH header, replace the all the bits in "traffic class", "flow label" and "hot limit" fields with "1".
- The first 8-bit "reserve" field with all "1"

TABLE VII
REGISTERS OF ALLREDUCE-SWITCH

Reg Name	Value	Default value
AGG_MODE	{31'h0, agg_mode}	agg_mode: 1'b0
AGG_PORT_CFG	{agg_port_num, 1'b0, agg_port_en}	agg_port_num: 5'h8 agg_port_en: 26'h000_00ff

Both CRC32 and CRC64 necessary for the variant length of frames. The polynomial used in RoCE v1 is $1+x^1+x^2+x^4+x^5+x^7+x^8+x^{10}+x^{11}+x^{12}+x^{16}+x^{22}+x^{23}+x^{26}+x^{32}$.

E. Output Stage

The major task in Output Stage is to output the frames from the aggregation flow and forwarding flow, which is handled by the Output Arbiter. Each Output Arbiter directly sends data to each 10GE TX interface. The function of Output Stage is similar to the function of Aggregation Output Stage, yet is much simpler. The data from the aggregation flow and all the branches from the forwarding flow are inquired with Round-Robin mechanism.

F. System Configuration

AllReduce-Switch offers the dynamic system configuration to flexibly meet the requirements in realistic applications. It provides two registers configured by the Serial Peripheral Interface (SPI) interface, which is listed in Tab. VII.

For *agg_mode*, 1'b1 is for floating-point addition, and 1'b0 is for fixed-point addition. *agg_port_num* indicates the port number for aggregation, and *agg_port_en* indicates the aggregate port bit by bit. In AllReduce-Switch, 2-8 ports are supported. The remaining bits are reserved for further extension. For example, if Ports 5-8 are used for aggregation, the value of *AGG_PORT_CFG* should be {5'h4, 1'b0, 26'h000_00f0}.

V. PERFORMANCE

In this paper, we have verified the performance of AllReduce-Switch in hardware implementation and distributed machine learning tasks. The acceleration on AllReduce communication further speeds up the training process.

A. Hardware Performance

Fig. 16 lists the hardware resources used in AllReduce-Switch with different port numbers. Note that the whole design is implemented according to 8 ports under the constraint of 200 MHz. Since it is trivial to separately implement the architecture with different numbers of ports, we estimate the hardware resource usage for other port numbers by subtracting the approximate consumption of the corresponding modules. Therefore, the curve is linear from 2 to 8 ports. Also note that the curve of utilization may be no longer linear when the design scales up, because the more complex cross-SLR design will introduce more logics.

As we analyze the challenges in Section III-B, BRAM resources become the major bottleneck in the implementation.

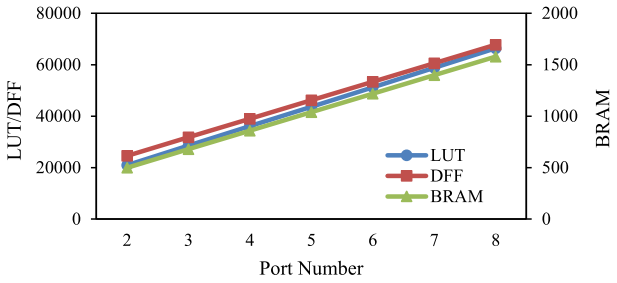


Fig. 16. Hardware resources usage in AllReduce-Switch with different port numbers.

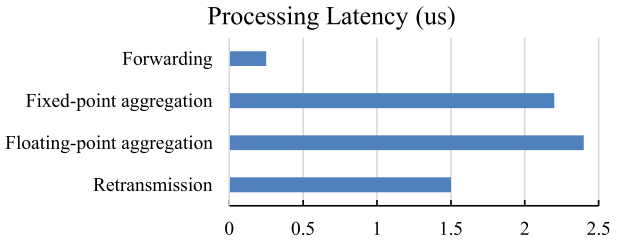


Fig. 17. Processing latency for different types of frames in AllReduce-Switch.

Xilinx Ultrascale FPGA series provide millions of Lookup Tables (LUTs) and D-type Flip-Flops (DFFs), but only several thousands of BRAMs. In XCVU190, the total 3780 BRAMs are evenly distributed in 3 SLRs. According to the physical layout, the top SLR and the bottom SLR are separated by the middle SLR. In our design, 6.18% of LUTs, 3.15% of DFFs and 41.75% of BRAMs are consumed, and 2 SLRs are used. Ideally, our design can be extended to over 16 ports, but the timing constraint is hard to be met. Therefore, in this case, data flows that operate in a single SLR, data flows that cross 2 SLRs and data flows that cross 3 SLRs coexist. Since the number of data flows squarely increase with the number of ports, it is extremely complex to do the cross-SLR design. The more plausible solution is to reduce the buffer size, but this requires the coordination of upper-layer applications.

The processing latency for different types of frames are shown in Fig. 17. The processing time for a typical frame is approximate to a constant, since the frame is not immediately processed in the switch. For example, the arbiter needs to routinely check the status of all the input flows. For different types of frames, the processing time is analyzed as follows:

- *Forwarding frames* go through Port Learning Stage, the first part of Frame Parsing Stage, and Output Stage. The latency is about 250 ns.
- *Aggregation frames* go through Port Learning Stage, Frame Parsing Stage, Aggregation Stage and Output Stage. The latency is about 2.2 us and 2.4 us for fixed-point aggregation and floating-point aggregation respectively.
- *Retransmission frames* go through Port Learning Stage, Frame Parsing Stage, the second part of Aggregation Stage and Output Stage. The latency is about 1.5 us.

TABLE VIII
THE EFFICIENT BANDWIDTH IMPROVEMENT IN ALLREDUCE OPERATION WITH DIFFERENT MODELS FOR 8 WORKERS COMPARED WITH RING-ALLREDUCE

Models	Model size (MB)	Ring-AllReduce comm. time (ms)	AllReduce-Switch comm. time (ms)	Time reduction ratio	Efficient BW improvement
AlexNet	236	322.9	184.4	42.89%	175.10%
GoogLeNet	51	70.0	39.9	43.00%	175.44%
ResNet-50	98	134.3	76.6	42.96%	175.32%
VGG-11	507	693.4	396.1	42.88%	175.06%
VGG-16	528	722.2	412.5	42.88%	175.08%
VGG-19	548	749.5	428.1	42.88%	175.08%

B. Performance in Distributed DNN Tasks

The performance of AllReduce-Switch is verified in distributed DNN training tasks on 2-8 workers respectively. Each worker equips an NVIDIA GeForce 2080 GPU, so that the multiple-machine single-GPU scenario applies. The SGD algorithm is used for gradient aggregation and the 32-bit floating point addition is used during the aggregation process. We compare the performance of AllReduce-Switch to that of the-state-of-the-art Ring-AllReduce algorithm implemented by OpenMPI and NCCL2. We do not quantitatively compare with other in-network aggregation frameworks, due to following reasons respectively: iSwitch targets to reinforcement learning with relatively small tensors instead of general DNNs [5]; SwitchML employs P4-programmable switching Application-Specific Integrated Circuit (ASIC) which we lack [14]; NetReduce targets to 100 Gbit/s communication scenarios that require for more memory on caching packets for streaming aggregation, so that the upper-layer setup is quite different [15].

First, we verify the efficient bandwidth improvement on AllReduce operation by using AllReduce-Switch. To accurately evaluate the communication time, we monitor the communication time in one iteration training instead of the whole training process. Not that the theoretical analysis in Section II only analyzes the communication time for data transmission. In fact, a round of AllReduce operation consists of not only the communication from data transmission, but also several times of extra end-to-end communication overhead like data preparation, *etc*.

Tab. VIII shows the communication time and the improvement ratio under the above setup with different models for 8 workers in AllReduce operation. 6 models including AlexNet [26], GoogLeNet [27], ResNet-50 [28], VGG-11, VGG-16 and VGG-19 [29] are verified on AllReduce-Switch for AllReduce operation. Let t_1 represent the communication time in Ring-AllReduce, and let t_2 represent the communication time in AllReduce-Switch. The reduction ratio is $\frac{t_1-t_2}{t_1}$, and the efficient bandwidth improvement is $\frac{t_1}{t_2}$, which are close to 0.429 and 1.75 respectively for all models. According to Tab. II, the efficient bandwidth improvement for NetReduce over Ring-AllReduce in communication time is $\frac{2(n-1)}{n}$. For 8 workers, the theoretical value is 1.75×. The improvement on efficient bandwidth is close to the theoretical value, since the data transmission dominates the AllReduce communication in the applications using 10GE interfaces. Similarly, we evaluate the efficient bandwidth on other numbers of workers, and the improvement in different models are close to each other, which are indistinguishable in Fig. 18.

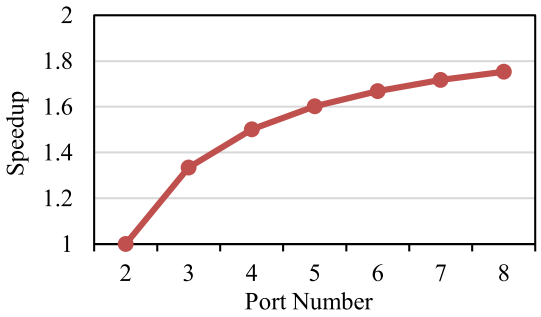


Fig. 18. The efficient bandwidth improvement with different numbers of workers using AllReduce-Switch compared with Ring-AllReduce.

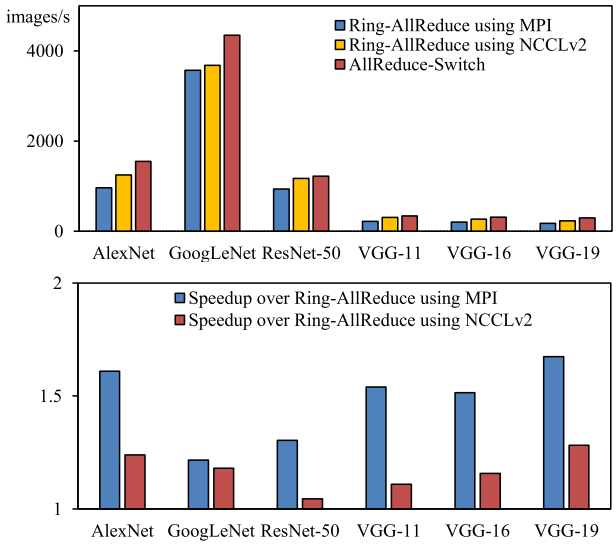


Fig. 19. Performance in image training tasks compared with Ring-AllReduce.

We further verify the performance of AllReduce-Switch on training throughput (*i.e.*, images per second), and compare the performance with Ring-AllReduce under the setting for 8 workers. Fig. 19 shows the comparison between AllReduce-Switch, Ring-AllReduce using MPI and NCCLv2 libraries. 6 models including AlexNet, GoogLeNet, ResNet-50, VGG-11, VGG-16 and VGG-19, with batch size of 128, 128, 64, 64 and 64 respectively, are verified in the image training tasks. The speedup ratio varies for different models, due to the different proportions of the communication part in the whole

training process. Setups like hardware setting and application software stack remain the same for different groups, so that we can credibly compare the performance.

MPI is originally used for High Performance Computing (HPC) and NCCL is specially designed for deep learning. As a consequence, NCCL is far more efficient at AllReduce operation for DNN training. Therefore, AllReduce-Switch improves MPI (1.22× to 1.67×) much better than NCCL (1.04× to 1.28×). It can be seen that for ResNet-50, NCCL is only improved by 4%. This is due to that ResNet-50 is a computing-intensive model in which the computation and the communication overlaps deeply. The benefits brought by AllReduce-Switch by reducing the communication part is hidden behind the computation part. In addition, we chose the maximum batch sizes of the models within the GPU memory limit. By reducing the batch size, the computation part accounts for smaller proportion, and then the speedup of AllReduce-Switch over Ring-AllReduce would be higher.

Note that previous works target to the framework, and optimization exists in all the aspects in the implementation of the system. In this paper, we focus on hardware design, and evaluate the hardware performance by eliminating the other differences from the baseline.

VI. Conclusion

In this paper, we present AllReduce-Switch, a scalable fully-pipelined architecture that supports in-network aggregated AllReduce communication. AllReduce-Switch maintains the maximum bandwidth, independent from the number of workers, while the efficient bandwidth of Ring-AllReduce drops with the increase of workers.

Since the NetReduce framework requires for the coordination between software and hardware, AllReduce-Switch is designed under typical specifications. The switch provides in-network aggregation to RoCE v1 frames that are recognized as aggregation frames, and hence performs NetReduce acceleration to AllReduce communication. Besides, forwarding frames are directly forwarded to the target ports, which is similar to the function of a normal network switch. In order to provide an efficient exception handling mechanism for the upper-layer applications, AllReduce-Switch buffers the aggregated frames and supports retransmission. The processing flow in AllReduce-Switch is separated with 4 stages: Port Learning Stage, Frame Parsing Stage, Aggregation Stage and Output Stage. Neighboring stages are separated with FIFOs, so that frames are efficiently processed in parallel in the form of pipeline. AllReduce-Switch is designed to support 8 workers with 10GE interfaces, and online configuration for the number of aggregation ports is supported. The design is implemented under the constraint of 200 MHz, and is ready to scale up to support more ports.

The data collected in machine learning tasks proves that the speedup ratio matches the theoretical analysis. This is because the communication time for data transmission dominates the AllReduce communication in the applications using 10GE interfaces. It also proves that AllReduce-Switch is a preferable choice in realistic applications with bandwidth limitations.

Acknowledgment

The authors would like to thank the important efforts from the editors and reviewers. They would also like to thank Abdurrashid Ibrahim Sanka for his contribution to this project.

References

[1] Y. Wu *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation," 2016, *arXiv:1609.08144*. [Online]. Available: http://arxiv.org/abs/1609.08144

[2] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, "Megatron-LM: Training multi-billion parameter language models using model parallelism," 2019, *arXiv:1909.08053*. [Online]. Available: http://arxiv.org/abs/1909.08053

[3] M. Li, "Scaling distributed machine learning with the parameter server," in *Proc. Int. Conf. Big Data Sci. Comput. (BigDataScience)*, 2014, pp. 583–598.

[4] S. Shi, X. Chu, and B. Li, "MG-WFBP: Efficient data communication for distributed synchronous SGD algorithms," in *Proc. IEEE Conf. Comput. Commun. (IEEE INFOCOM)*, Apr. 2019, pp. 172–180.

[5] Y. Li, I.-J. Liu, Y. Yuan, D. Chen, A. Schwing, and J. Huang, "Accelerating distributed reinforcement learning with in-switch computing," in *Proc. 46th Int. Symp. Comput. Archit. (ISCA)*, Jun. 2019, pp. 279–291.

[6] B. Zhang, L.-Y. Chen, and N. Verma, "Neural network training with stochastic hardware models and software abstractions," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 68, no. 4, pp. 1532–1542, Apr. 2021.

[7] A. A. Awan, C.-H. Chu, H. Subramoni, and D. K. Panda, "Optimized broadcast for deep learning workloads on dense-GPU InfiniBand clusters: MPI or NCCL?" in *Proc. 25th Eur. MPI Users' Group Meeting*, Sep. 2018, pp. 1–9.

[8] R. Kobus, D. Jünger, C. Hundt, and B. Schmidt, "Gossip: Efficient communication primitives for multi-GPU systems," in *Proc. 48th Int. Conf. Parallel Process.*, Aug. 2019, pp. 1–10.

[9] A. Li *et al.*, "Evaluating modern GPU interconnect: PCIe, NVLink, NV-SLI, NVSwitch and GPUDirect," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 1, pp. 94–110, Jan. 2020.

[10] D. Foley and J. Danskin, "Ultra-performance pascal GPU and NVLink interconnect," *IEEE Micro*, vol. 37, no. 2, pp. 7–17, Mar./Apr. 2017.

[11] R. Mayer and H.-A. Jacobsen, "Scalable deep learning on distributed infrastructures: Challenges, techniques, and tools," *ACM Comput. Surv.*, vol. 53, no. 1, pp. 1–37, May 2020.

[12] A. Gibiansky. (2017). Bringing HPC Techniques to Deep Learning. [Online]. Available: https://andrew.gibiansky.com/blog/machine-learning/baidu-allreduce/

[13] R. L. Graham *et al.*, "Scalable hierarchical aggregation and reduction protocol (SHARP) streaming-aggregation hardware design and evaluation," in *Proc. Int. Conf. High Perform. Comput.* Cham, Switzerland: Springer, 2020, pp. 41–59.

[14] A. Sapio *et al.*, "Scaling distributed machine learning with in-network aggregation," 2019, *arXiv:1903.06701*. [Online]. Available: http://arxiv.org/abs/1903.06701

[15] S. Liu *et al.*, "NetReduce: RDMA-compatible in-network reduction for distributed DNN training acceleration," 2020, *arXiv:2009.09736*. [Online]. Available: http://arxiv.org/abs/2009.09736

[16] L. Wang, X. Zhao, D. Kaeli, Z. Wang, and L. Eeckhout, "Intra-cluster coalescing and distributed-block scheduling to reduce GPU NoC pressure," *IEEE Trans. Comput.*, vol. 68, no. 7, pp. 1064–1076, Jul. 2019.

[17] T. Chilimbi, Y. Suzue, J. Apacible, and K. Kalyanaraman, "Project adam: Building an efficient and scalable deep learning training system," in *Proc. 11th USENIX Symp. Operating Syst. Design Implement. (OSDI)*, 2014, pp. 571–582.

[18] C. Guo *et al.*, "RDMA over commodity Ethernet at scale," in *Proc. ACM SIGCOMM Conf.*, Aug. 2016, pp. 202–215.

[19] Z. Guo and Y. Yang, "On nonblocking multicast fat-tree data center networks with server redundancy," *IEEE Trans. Comput.*, vol. 64, no. 4, pp. 1058–1073, Apr. 2015.

[20] A. Sergeev and M. Del Balso, "Horovod: Fast and easy distributed deep learning in TensorFlow," 2018, *arXiv:1802.05799*. [Online]. Available: http://arxiv.org/abs/1802.05799

[21] Y. Ueno and R. Yokota, "Exhaustive study of hierarchical AllReduce patterns for large messages between GPUs," in *Proc. 19th IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput. (CCGRID)*, May 2019, pp. 430–439.

[22] P. Wang, G. Wen, X. Yu, W. Yu, and T. Huang, "Synchronization of multi-layer networks: From node-to-node synchronization to complete synchronization," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 66, no. 3, pp. 1141–1152, Mar. 2019.

[23] P. Bosshart *et al.*, "Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 99–110, Oct. 2013.
[24] Z. Guo and Y. Yang, "High-speed multicast scheduling in hybrid optical packet switches with guaranteed latency," *IEEE Trans. Comput.*, vol. 62, no. 10, pp. 1972–1987, Oct. 2013.
[25] InfiniBand Trade Association. (2015). *Infiniband Architecture. Specification Volumen 1. Release 1.3*. [Online]. Available: http://www.infinibandta.org
[26] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.
[27] C. Szegedy *et al.*, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 1–9.
[28] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.
[29] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*. [Online]. Available: http://arxiv.org/abs/1409.1556

Yao Liu received the B.S. and M.S. degrees in microelectronics from Fudan University, Shanghai, China, in 2008 and 2011, respectively, and the Ph.D. degree from the Department of Electrical Engineering, City University of Hong Kong (CityU), Hong Kong, in 2019. He is currently holding a post-doctoral position with the CityU Architecture Lab for Arithmetic and Security (CALAS) Group. Before he joined CityU, he worked in the IC industry as a VLSI engineer for several years. His research interests include FPGA prototyping and VLSI implementation for hardware security systems, computer architecture, and network systems.

Junyi Zhang received the B.Eng. degree in electrical and computer engineering from the University of Science and Technology of China, Hefei, China, in 2013, and the M.S. degree in electrical and computer engineering from Rice University, Houston, USA, in 2016. He is currently a Senior Engineer with HUAWEI 2012 Labs. His research interests include distributed systems, datacenter networks, and FPGA-based system design.

Shuo Liu received the B.Eng. degree in electrical and computer engineering from Harbin Institute of Technology, Harbin, China, in 2012, and the Ph.D. degree in electrical and computer engineering from the National University of Singapore in 2017. He was a Research Fellow with Berkeley Education Alliance for Research in Singapore in 2017. He is currently a Principle Engineer with HUAWEI 2012 Labs. His research interests are in control and optimization for robust parallel and distributed systems.

Qiaoling Wang received the B.Eng. degree in automation and control from Harbin University of Science and Technology, Harbin, China, in 2004, and the Ph.D. degree in automation and control from Harbin Institute of Technology, Harbin, in 2010. She is currently a Senior Expert with HUAWEI 2012 Labs. Her research interests lie in computer networking, especially in in-network computing for high-performance computing and distributed machine learning.

Wangchen Dai received the B.Eng. degree in electrical engineering and automation from Beijing Institute of Technology, China, in 2010, the M.A.Sc. degree in electrical and computer engineering from the University of Windsor, Canada, in 2013, and the Ph.D. degree in electronic engineering from the City University of Hong Kong in 2018. After completing the Ph.D. study, he had appointments at Hardware Security Lab, Huawei Technologies Company Ltd., in 2018, and the Department of CSSE, Shenzhen University in 2020, respectively. He is currently working as a Senior Security Researcher with Security Team, ByteDance Ltd., Shenzhen. His research interests include cryptographic hardware and embedded systems, fully homomorphic encryption, and reconfigurable computing. He received the National Natural Science Foundation of China (Youth Program) in 2020.

Ray Chak Chung Cheung (Member, IEEE) received the B.Eng. (Hons.) and M.Phil. degrees in computer engineering and computer science and engineering from The Chinese University of Hong Kong (CUHK) in 1999 and 2001, respectively, and the D.I.C. and Ph.D. degrees in computing from Imperial College London (IC) in 2007. After completing his Ph.D. study, he received the Hong Kong Croucher Foundation Fellowship and moved to Los Angeles, at the Electrical Engineering Department, UCLA, where he spent two years with the Image Communication Lab for continuing his research work. He is currently an Associate Professor with the Department of Electrical Engineering, City University of Hong Kong, and the Digital Systems Lab. His current research interests include cryptographic hardware and embedded system designs.