2021 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)

Co-designing the Topology/Algorithm to Accelerate Distributed Training

Xiang Hou, Rui Xu, Sheng Ma, Qiong Wang, Wei Jiang, Hongyi Lu
Science and Technology on Parallel and Distributed Processing Laboratory, College of Computer
National University of Defense Technology
Changsha, China
{houxiang, xurui16a, masheng, wangqiong, jiangwei, hylu}@nudt.edu.cn

***Abstract*—With the development of Deep Learning (DL), Deep Neural Network (DNN) models have become more complex. At the same time, the development of the Internet makes it easy to obtain large data sets for DL training. Large-scale model parameters and training data enhance the level of AI by improving the accuracy of DNN models. But on the other hand, they also present more severe challenges to the hardware training platform because training a large model needs a lot of computing and memory resources that can easily exceed the capacity of a single processor. In this context, integrating more processors on a hierarchical system to conduct distributed training is a direction for the development of training platforms. In distributed training, collective communication operations (including all-to-all, all-reduce, and all-gather) take up a lot of training time, making the interconnection network between computing nodes one of the most critical factors affecting the system performance. The hierarchical torus topology, combined with the Ring All-Reduce collective communication algorithm, is one of the current mainstream distributed interconnection networks. However, we believe that its communication performance is not the best. In this work, we first designed a new intra-package communication topology, i.e. the switch-based fully connected topology, which shortens the time consumed by cross-node communication. Then, considering the characteristics of this topology, we carefully devised more efficient all-reduce and all-gather communication algorithms. Finally, combined with the torus topology, we implemented a novel distributed DL training platform. Compared with the hierarchical torus, our platform improves communication efficiency and provides 1.16-2.68 times speedup in distributed training of DNN models.**

***Index Terms*—topology, hardware training platform, distributed training, collective communication**

I. Introduction

Today, AI profoundly affects our daily lives, playing an inestimable role in a wide range of applications such as robotics, language recognition [9] and image recognition [13], [28], [34]. The DNN model is one of the most important methods to realize AI. To quickly training DNN models, architecture researchers constantly seek to invent more effective accelerators, e.g. GPU [16], TPU, FPGA, etc. In recent years, with the rapid development of the Internet, large data sets for DL training can be easily obtained. For example, the ImageNet [10] data set used for object recognition training contains about 1.28 million images. At the same time, to improve the ability of DL, the number of neural network layers has increased from the original 5 layers to hundreds of layers today. The growth of the data set size and the network layers make the training process of DL consume a lot of memory and computing resources. In this context, the continuous use of a single accelerator for training faces the problem of low efficiency, which is far from the needs of academia and industry.

This work is supported in part by the Science and Technology Innovation project of Hunan Province No. 2018RS3083, in part by the National Key Research and Development Project No. 2018YFB0204301. Sheng Ma and Qiong Wang are the corresponding authors.

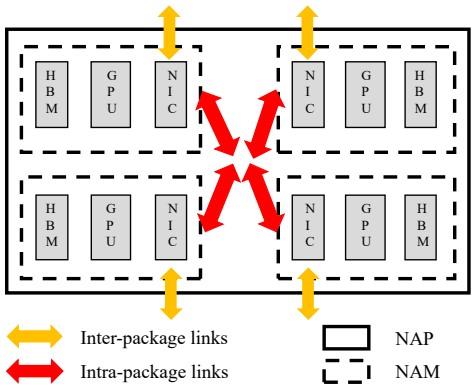


Fig. 1. Distributed DL hardware training platform

Using distributed DL training platforms to assign training tasks on multiple processors is a way to improve the speed of DNN training. The distributed platform divides data and models into multiple processors and performs training tasks together, which greatly reduces the memory and computing ability requirements of a single processor. Moreover, through the cooperation training of all processors, the time required to train the DNN model will be greatly shortened.

Figure 1 shows the architecture of a typical distributed platform [11], [17], which is a hierarchical structure composed of multiple Neural Accelerator Packages (NAP) [24]. And each NAP integrates with multiple Neural Accelerator Modules (NAM), which consists of a computing node, a high-bandwidth memory (e.g. HBM) module, and a dedicated network interface card. Among them, the HBM module is used to store DNN models and training data, the computing node performs matrix multiplication and addition operations in DNN training,

978-0-7381-2646-3/21/$31.00 ©2021 IEEE
DOI 10.1109/ISPA-BDCloud-SocialCom-SustainCom52081.2021.00141

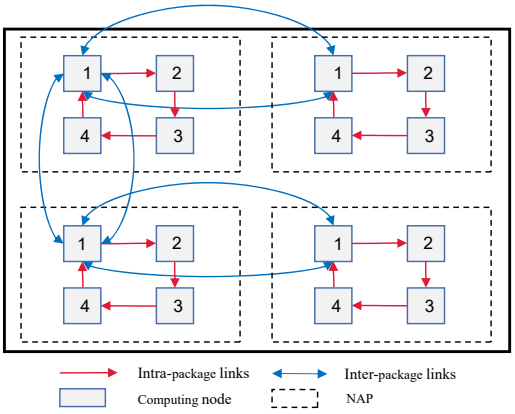


Fig. 2. The hierarchical torus topology. For inter-package links, the figure only shows the connections between all nodes labeled 1 in different NAPs.

and the NIC is responsible for communication operations. Besides, the hierarchical interconnection network topology comprises the high bandwidth intra-package links (e.g. PCI-E) for NAM-to-NAM communication and the inter-package links (e.g. Ethernet [3]) for NAP-to-NAP communication [24]. In this distributed training system, data communication between computing nodes consumes a lot of training time. Therefore, to improve system efficiency, it is necessary to study the structure of the hierarchical topology.

In this work, by optimizing the hierarchical torus topology (Figure 2) [24], we present a novel interconnection network of distributed DL training platform. In the hierarchical torus topology, the local dimension is a ring structure, which is not friendly to cross-node communication. Therefore, the deployment of a fully connected structure is what we expect. Furthermore, in the Ring All-Reduce [19] collective communication algorithm, the feature that there is only communication between adjacent nodes makes it possible to achieve good performance in a ring topology. However, the algorithm cannot make full use of the links in the fully connected topology. Therefore, designing more efficient all-reduce and all-gather collective communication algorithms is the second focus of the research.

Specifically, by i) modifying the intra-package communication topology of the hierarchical torus, and ii) designing unique collective communication algorithms for the new topology, we improve the communication efficiency between nodes in the distributed platform, and finally, accelerate the training of the DNN models. In particular, we make the following contributions:

- For the intra-package interconnection structure of the distributed platform, we propose the switch-based fully connected topology, that is, by using a few local-switches to connect computing nodes, which significantly improves the speed of the cross-node communication.
- We propose more efficient all-reduce and all-gather collective communication algorithms. The communication involved in these two algorithms can make full use of the links provided by the fully connected topology, which reduces the steps and time of communication compared to the Ring All-Reduce algorithm.
- On our proposed platform, we simulate the distributed training of three DNN models. And the results prove that our work builds an efficient distributed DL training system. Compared with the hierarchical torus, it greatly improves the speed of training.

The rest of the paper is organized as follows. In Section II, we introduce some background and works related to the distributed training and collective communication algorithms. Section III describes our proposed distributed DL hardware training platform. The experimental method used to evaluate the platform and the results are described in Section IV and Section V, respectively. Finally, we conclude the paper in Section VI.

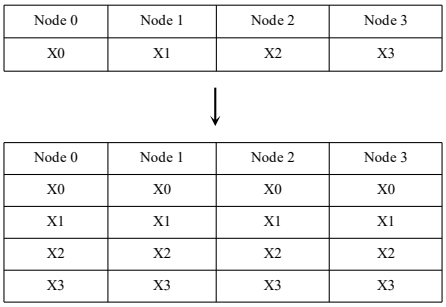
II. Background and Related Work

A. Distributed Training

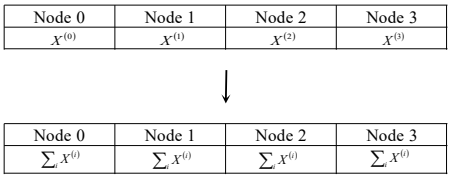
In order to cope with the challenges brought by the surge in model parameters and training data, researchers have begun to conduct distributed training on multi-processor platforms. Distributed training refers to splitting the DNN model and/or training data among multiple processors to reduce the computational pressure of each processor.

When multiple processing nodes are used to perform training tasks in parallel, there are two ways to update the weights of the model, i.e. asynchronous [4], [36], [37] and synchronous [5], [8], [27], [35]. Asynchronous updates are usually used in the parameter server framework. In which, the training data is distributed over worker nodes, and the server node is responsible for collecting the gradient of some nodes and updating their model through the reduction operation [15]. Asynchronous updates will generate the stale gradients [5], i.e. the model may have been updated while a worker is computing its local gradient, which reduces the training accuracy. Therefore, the current mainstream researches adopt synchronous updates. That is, each node, firstly, generates a local gradient through calculation and then reduces the local gradient on all nodes. Lastly, before the start of the next iteration, each node updates its model parameters at the same time. To ensure training accuracy, our work is also based on synchronous updates.

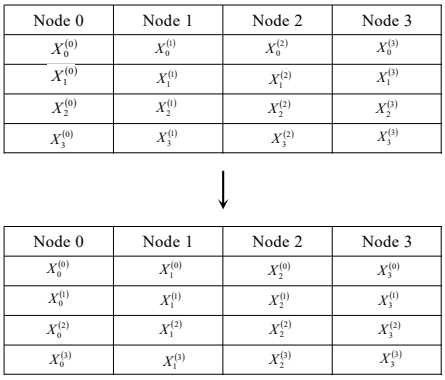
The main parallel strategies currently used are: data parallelism [1], [14], [30], [33], model parallelism [6], [13] and hybrid parallelism [12], [32]. In data parallelism, the training data is first segmented and allocated to different computing nodes (each node has a complete DNN model) for training to generate the local gradient. Then the local gradients of multiple nodes are communicated to generate the final gradient, and finally, the weights are updated [26]. In model parallelism, the DNN model is segmented and assigned to different nodes (each node will process all training data). Due to the segmentation of the DNN model, communication between multiple nodes is also required to generate the final



(a) All-gather



(b) All-reduce



(c) All-to-all

Fig. 3. Collective communication operations in distributed training

gradient and update the weights. In hybrid parallelism, all nodes are divided into different groups, and different parallel strategies are used within or between groups.

B. Collective Communication Operations

To realize distributed training on the hardware platform, the communication of three types of data, i.e. activations, input gradients, weight gradients, between different nodes is inseparable in the three stages of training, i.e. forward propagation (FP), backward propagation (BP), weight update (WU). And the communication of the data is completed by collective communication operations. Collective communication refers to multiple nodes simultaneously communicating data and performing specific operations [24]. Our work will cover all three kinds of operations in distributed training, i.e. all-gather, all-reduce, all-to-all. And Figure 3 uses four nodes as an example to show the initial and final states of the communication.

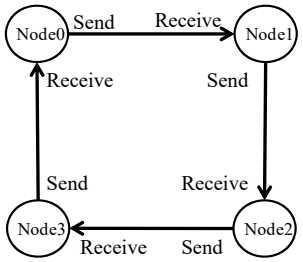


Fig. 4. Logical structure of the Ring All-Reduce algorithm

In distributed training of DNN, all-gather (Figure 3(a)) and all-reduce (Figure 3(b)) are the most common collective communication operations. Specifically, through all-gather, each node will eventually get the data of any other node. In model parallelism, the communication of activations in the FP process is completed by all-gather. All-reduce is similar to all-gather. But after getting the data, it will perform the reduction operation. Therefore, the all-reduce operation is usually performed in the WU phase of data parallelism. At present, the Ring All-Reduce [19] is the most popular all-reduce algorithm. Figure 4 graphically depicts the basic logical structure of the algorithm. In this algorithm, each node can only receive data from the previous node and send data to the next node, which logically forms a ring. Although the Ring All-Reduce algorithm can be applied to any physical topology without any modification, in some cases, e.g. fully connected topology, the link utilization rate of this algorithm is extremely low.

In terms of the all-to-all operation, each node needs to send different parts of the data to other nodes. When using model parallelism to train the embedding layer of DLRM [18], the all-to-all operation is indispensable in the FP and BP stages. Figure 5 shows the most widely used all-to-all communication algorithm described in [24], which is composed of $N-1$ steps (N is the number of computing nodes). The data of each node is divided into N messages. During the ith step, the node sends data to the node with the distance of i and receives data from the node with the distance of i. We can see that this algorithm involves communication between non-adjacent nodes (e.g. Figure 5(c)). However, in the hierarchical torus topology (Figure 2), the structure of the local dimension is a ring (solid red lines), which is not friendly to cross-node communication. Therefore, we need to study a topology that allows direct communication between arbitrary nodes.

C. Related Work

With the increasing complexity of DNN models and the continuous emergence of training data, researchers are increasingly exploring the distributed training platforms. Based on the GARNET network simulator [2], the authors in [24] released a

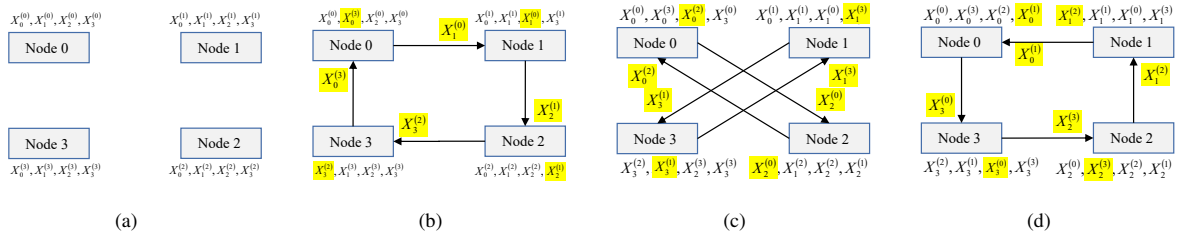


Fig. 5. All-to-all collective communication algorithm

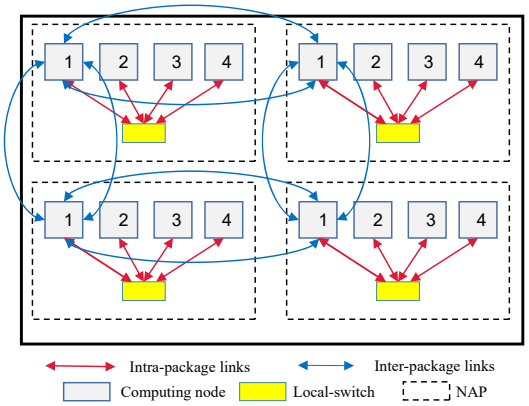


Fig. 6. Proposed distributed DL training topology

distributed training platform simulator that can collaboratively design collective communication algorithms and topologies, and conducted in-depth research on the intra-package networks. And the authors in [22] connected the ASTRA-sim DNN training simulator [24] and the NS3 simulator [25] to supply a facility that can analyze the inter-package networks. However, these two works only focus on the hierarchical torus and alltoall topologies. Although these two topologies have high scalability, the training efficiency is not the best.

In addition, to improve the speed of all-reduce operations, the authors in [20] optimized the Ring All-Reduce algorithm based on tree topology. And to increase the utilization of the high bandwidth provided by intra-package links, the authors in [23] proposed the Accelerator Collectives Engine (ACE) microarchitecture. By embedding the ACE into the NIC, the distributed training speed is greatly accelerated. However, all the above works use the Ring All-Reduce algorithm or its variants, which is not suitable for the fully connected topology we proposed. The authors in [31] proposed the Recursive Doubling algorithm to improve the collective communication operation of short or medium-sized messages ($< 512KB$), but this algorithm is not suitable for long messages, and its efficiency is not even as good as the ring algorithm [31]. Therefore, by comparison, the collective communication algorithm we proposed is suitable for distributed training of DNN models of any scale.

III. Proposed Distributed DL Training Platform

A. Proposed Topology

As mentioned in Section II-B, the all-to-all collective communication algorithm requires direct communication between any nodes. Taking N nodes as an example, based on the fully connected topology, the communication can be completed quickly within $N-1$ steps. However, if a unidirectional ring topology is used, the steps will be greatly increased, i.e. from $N-1$ to $\frac{N\times(N-1)}{2}$. Therefore, we need to present a more suitable topology other than the unidirectional ring topology.

1) Overview: Figure 6 demonstrates the proposed distributed DL training topology. In the intra-package topology, we use the switch-based fully connected topology to interconnect local computing nodes through high bandwidth intra-package links and some local-switches. Intuitively, compared to the ring topology, this topology provides a direct data path between any nodes. The experimental results described in Section V-A prove that our topology greatly improves the speed of all-to-all communication.

In the inter-package topology, we use the torus topology as the existing work [7]. As depicted in Figure 6, the computing nodes with the same ID in different NAMs are connected by inter-package links, and these lines form bidirectional rings in the horizontal and vertical dimensions [24]. This part of the topology will be the focus of our next phase of research. Overall, we propose a hierarchical topology for distributed training platforms.

2) Local-switch: To reduce the time consumed by all-to-all collective communication operations, the physical links for direct communication between all nodes within a NAP are expected. So, the naive fully connected topology is one of our choices. However, as the number of nodes increases, the on-chip resources occupied by this structure will increase exponentially, which not only wastes precious on-chip space but also has poor scalability. Besides, in each step of the collective communication operations, each node only needs to communicate with at most two nodes, which makes the link utilization rate of naive fully connected topology extremely low. Considering these shortcomings, we propose to use the local-switch to achieve full connectivity between nodes.

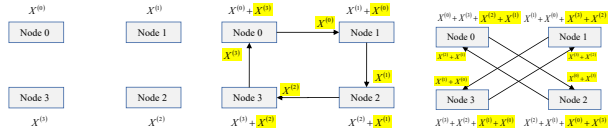


Fig. 7. Proposed all-reduce algorithm

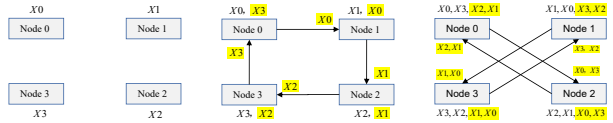


Fig. 8. Proposed all-gather algorithm

In our topology, the local-switch is a high-bandwidth switch used for intra-package communication. Each of them has a certain number of high-bandwidth link ports and a $N \times N$ fully connected crossbar used to provide a direct data path between different computing nodes. For example, in Figure 6, through a four-port local switch, any two nodes within a NAP can communicate directly through one hop, i.e. using a local-switch can already provide fully connected features. Even so, if all communication operations between nodes are completed through one local-witch, it will cause congestion in the network and cannot bring the expected performance improvement.

In order to maximize the acceleration of collective communication operations, it is desirable to use N local-switches in the NAP containing N nodes, to ensure that in each communication step, there is an idle local-switch between any two nodes that need to communicate. However, if too many local-switches are used, a lot of power and area will be consumed. This forces us to make a trade-off between performance and consumption. We will analyze this issue in detail in the experimental section.

B. Proposed Collective Communication Algorithms

The topology proposed in Section III-A, different from the ring structure, provides an opportunity for direct data exchange between any computing nodes within a NAP. However, in the Ring All-Reduce algorithm, there is no such communication. To take advantage of these fully connected links, we design new all-reduce and all-gather collective communication algorithms. Figures 7 and 8 respectively describe the steps of these two algorithms in the case of N nodes (here N = 4).

1) All-reduce: As shown in Figure 7, the all-reduce operation takes $\log_2 N$ steps. During the first step, node i sends data $X^{(i)}$ (yellow part) to the neighboring node and receives corresponding data from its previous node. After communication, it adds the received data with its local data (Figure 7(b)). Then, during the Sth step (S is greater than or equal to 2), the node i sends its data to the $(i + 2 \times (S-1))$ mod N node and receives corresponding data.

2) All-gather: Figure 8 shows that the all-gather operation also takes $\log_2 N$ steps. However, it does not contain any local reduction operations. During the Sth step, each node only needs to send all its data to the receiving node.

3) Analysis: Compared with the Ring All-Reduce algorithm, our proposed algorithm does not need to divide the data owned by each node and allows directly one-hop communication between any nodes, e.g. node 0 to node 2. Therefore, combined with the fully connected topology proposed in Section III-A, our algorithm significantly reduces the communication steps.

C. Summary of All Communications on the Proposed Platform

During the training process, all communications between computing nodes will be carried out in the order of local, vertical and horizontal dimensions. The data of each communication contains all the data required by the intermediate nodes and the final node. In the local dimension, data is communicated according to the steps of the algorithm proposed in Section III(B). In horizontal and vertical dimensions, the platform executes the collective communication algorithms according to the [19], i.e. the Ring All-Reduce algorithm. In all dimensions, when all-to-all communication operations are required, the algorithm process introduced in Section II-B is the flow of data.

In summary, through the co-design of topology and collective communication algorithm, we optimize the intra-package interconnection network of the hierarchical torus topology and construct a distributed DL training platform that can efficiently train complex DNN models.

TABLE I
WORKLOADS

Number	Workload	layers	Parallel strategy	Collective communication operation
1	MicroAllReduce (512KB, 1MB, 2MB, 4MB)	1	None	All-reduce
2	MicroAllToAll (1MB)	1	None	All-to-all
3	DLRM_Hybrid Parallel	8	Hybrid	All-to-all + All-reduce
4	MLP_ModelParallel	6	Model	All-gather + All-reduce
5	MLP_HybridParallel_Data_Model	6	Hybrid	All-gather + All-reduce

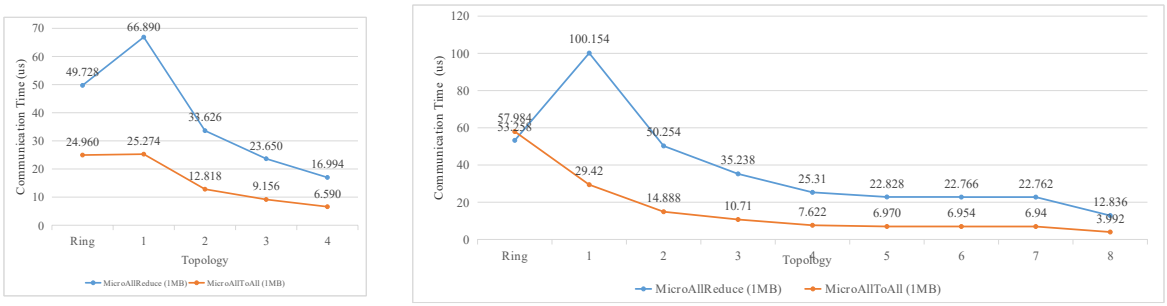
IV. Experimental Methodology

A. Experimental Tools

We use the following tools to explore the platforms proposed in this paper:

TABLE II
SIMULATION CONFIGURATIONS

Parameter	Value	
	Proposed platform	Hierarchical torus
Number of nodes within a NAP	4 or 8	
Number of NAPs	1 or 4	
Numbers of local-switches within a NAP	1 to N	None
VC/VNET	2	
Buffers per VC	5000	
Intra-package link width	512bits/ns	
Inter-package link width	256bits/ns	
Router latency	5ns	
Intra-package link latency	90ns	
Inter-package link latency	200ns	
Cycle period	1ns	



(a) (b)

Fig. 9. The time of all-to-all and all-reduce collective communication operations in different topologies with 4 NAMs (a) or 8 NAMs (b).

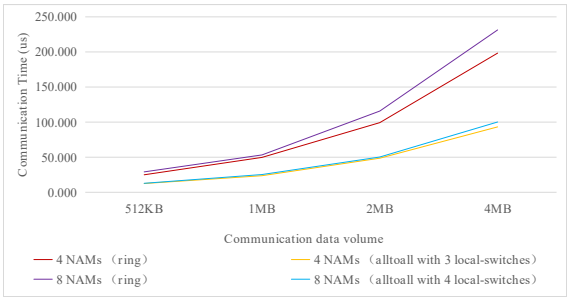


Fig. 10. The communication time required to complete the all-reduce operation with different communication data sizes

1) Performance Evaluation Tool: The ASTRA-sim [24] is a distributed DL training platform simulator. The simulator can simulate cycle-level communication behavior for the parallel strategies including data parallelism, model parallelism, and hybrid parallelism. We modify it to support our proposed hierarchical topology and collective communication algorithms.

2) Area and Power Consumption Evaluation Tool: DSENT is a widely used network-level modeling framework for electronic NoC, with integrated timing, area, and power models, and these models are accurate in the deep sub-100 nm regime [29]. So, we use DSENT to evaluate the power and area of different topology settings in this work.

B. Experimental Parameters

1) Workloads: The workloads used in the experiments are given in Table I. The first two workloads have only one layer and are used to only analyze the all-to-all and all-reduce collective communication algorithms (the data in brackets indicates the size of the communication data volume). The remaining three workloads are complete DNN models to study all three parallel strategies and three collective communication operations mentioned above.

The workload, in addition to the network model, also includes the calculation time and the amount of data transfer required for each layer of the neural network calculated based on some data sets using an DNN accelerator simulator [21]. In the third workload, the embedding layer uses model parallelism (due to a large number of model parameters), and the remaining 7 layers (conventional MLP layer) use data parallelism. In the fifth workload, the data parallelism is used across NAPs, and the model parallelism is used across nodes within a NAP.

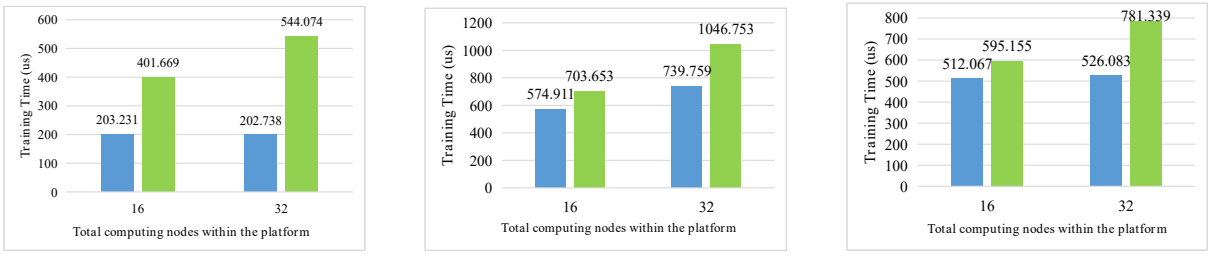
2) Simulated Hardware distributed platform configurations: In this work, we respectively apply our proposed platform and the hierarchical torus to explore the performance of distributed training of the DNN models. Table II shows the main simulation configurations of the platforms, routers, and links. The number *N* in the *Numbers of local-switches within a NAP* row indicates how many computing nodes are in a NAP. In the *Number of NAPs* row, the platform composed of one NAP is mainly used to study the efficiency of all-to-all and all-reduce collective communication algorithms. And four NAPs are used to compare the training time of the three complete DNN models.

V. Results

The content of this section is arranged as follows. In Section V-A, we compare the speed of the collective communication operations in the switch-based fully connected topology and the ring topology. And we analyze the trade-offs between performance and power consumption when using different numbers of local-switches in a fully connected topology. In Section V-B, we use three complete DNN models to evaluate our distributed DL training platform. Lastly, we use DSENT to measure the hardware overhead of the platforms in Section V-C.

A. Switch-based Fully Connected Topology vs. Ring Topology for All-to-all and All-reduce Operations

To verify the effect of our proposed topology and all-reduce algorithm on accelerating the collective communication operations, we conduct experiments on the platform composed of 1 NAP with 4 or 8 NAMs. The workloads are *MicroAllReduce (1MB)* and *MicroAllToAll (1MB)*, and each experiment includes only one iteration for simplicity. Also, to analyze the impact of the number of local-switches on the performance of the fully connected topology, we simulate our platforms with



(a) (b) (c)

Fig. 11. The time consumed by the hardware platforms to train the DNN models. Including DLRM_Hybrid Parallel (a), MLP_ModelParallel (b) and MLP_HybridParallel_Data_Model (c). The blue bar represents the switch-based fully connected platform we proposed, and the green bar represents the hierarchical torus platform.

2 to N local-switches. The experimental results are shown in Figure 9(a) and 9(b), from which the following analysis can be made.

Firstly, compared with the ring topology, our fully connected topology accelerates the all-to-all operation. When 4 and 8 NAMs are set, the communication time is reduced by up to 73.59% and 93.11%, respectively. In the latter comparison, this substantial acceleration comes from that the characteristics of the fully connected topology make the all-to-all communication algorithm save 21 steps.

Secondly, for the all-reduce operation, our proposed algorithm allows the data to reach the end node directly. And combined with the fully connected topology, the communication steps are reduced. When 4 and 8 NAMs are set, the communication time is decreased by 65.82% and 75.89% respectively at most. In comparison, our proposed topology is more suitable for all-to-all collective communication algorithm.

Thirdly, it can be seen from the graph that, in these two configurations of fully connected topologies, setting three and four local-switches respectively can bring higher communication speeds. If we continue to increase the number of local-switches, the hardware overhead will also increase, which will bring a poorer price-performance ratio. Therefore, in subsequent experiments, we only conduct an in-depth analysis of the topology with three or four local-switches embedded in each NAP.

Further, for the all-reduce operation, we analyze different communication data sizes, and the results are shown in Figure 10. It can be observed that the method proposed in this paper is always better than the ring structure, which proves that our platform can train large Deep Neural Networks.

B. Proposed Platform vs. Hierarchical Torus for the Training Time of DNN Models

Figure 11 describes the training time consumed when training different DNN models, including two training iterations, on the two kinds of platforms (3 or 4 local-switches within a NAP in our platforms). Each platform consists of 4 NAPs with 4 (the left side of each subfigure) or 8 (the right side) computing nodes per NAP. By comparing the training time on different platforms, we can draw the following conclusions.

Firstly, compared with the hierarchical torus, our platform significantly improves the distributed training speed of all three DNN models. In this work, a speedup of 1.16-2.68 times is achieved. This is because, on a distributed platform, the training process relies on the support of collective communication operations. Therefore, optimizing the interconnection network can greatly improve training efficiency.

Secondly, the most noticeable acceleration appears on the right side of figure 11(a). Besides, among all three workloads, the time consumed to train the *DLRM_Hybrid Parallel* model is reduced the most, which once again confirms that our fully connected topology is more suitable for all-to-all collective communication operations.

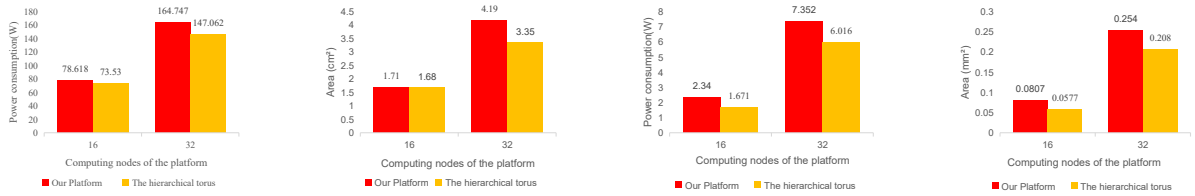
Finally, when more computing nodes are set up in a NAP, i.e. from four to eight, the acceleration provided by our platform is even more significant. For example, in figure 11(b), the speedup increases from 1.22× to 1.41×. It means that our platform also has extremely high scalability, which is particularly important when larger-scale distributed systems need to be deployed.

C. Power Consumption and Area Analysis

To evaluate the hardware overhead, we use DSENT to analyze the power and area consumed by the links and routers in our proposed topology and the hierarchical torus with 45 nm bulk LVT. And the result is shown in Figure 12. The power depicted in Figure 12(a) and 12(c) includes dynamic power and static power.

Because our topology uses some additional structures, it inevitably consumes more resources. In the experiment, when there are four computing nodes in each NAP, our topology consumes 7.65% more energy than the hierarchical torus topology and takes up 1.78% more area. If there are eight nodes in each NAP, compared with the hierarchical torus, the increased energy consumption is 12.42%, and the increased area occupied is 25.07%.

Because there are many other functional modules in a distributed system, such as computing units, storage units, etc., the interconnection network is only a small part of the platform. Therefore, given the obvious performance improvement, we think this small additional on-chip overhead is worthwhile.



(a) Router power consumption (b) Router area consumption (c) Link power consumption (d) Link area consumption

Fig. 12. The power and area consumption of routers and links within different platforms

VI. Conclusion

In recent years, with the continuous development of DL and the Internet, DNN models have become more and more complex, and at the same time, more and more training data have been brought, which lead to an exponential increase in training time. To meet this challenge, research on distributed DL training platforms is one of the strategies.

In this work, we first lay out an intra-package communication topology, i.e. the switch-based fully connected topology, which accelerates the all-to-all collective communication operation. Then, based on this topology, unprecedented all-reduce and all-gather algorithms are proposed to accelerate the collective communication operations commonly used in DNN training. In other words, by combining the research on hierarchical topology and collective communication algorithm, we optimize the hierarchical torus interconnection network and eventually build a distributed DL training platform that can greatly improve the training efficiency.

In future work, we plan to use a larger-scale training platform as the background to conduct a more in-depth discussion on the number of local-switches within a NAP. And we try to design a new inter-package communication topology, hoping to further reduce the communication time consumed when training DNN models.

References

[1] Tensorflow: Large-scale machine learning on heterogeneous distributed systems. 2016.

[2] N. Agarwal, T. Krishna, L. S. Peh, and N. K. Jha. Garnet: A detailed on-chip network model inside a full-system simulator. In *IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS 2009, April 26-28, 2009, Boston, Massachusetts, USA, Proceedings*, 2009.

[3] M. Beck and M. Kagan. Performance evaluation of the rdma over ethernet (roce) standard in enterprise data centers infrastructure. *International Teletraffic Congress*, 2011.

[4] S. Chaturapruek, J. C. Duchi, and Christopher RXe. Asynchronous stochastic convex optimization: the noise is in the noise and sgd don't care. 2015.

[5] J. Chen, R. Monga, S. Bengio, and R. Jozefowicz. Revisiting distributed synchronous sgd. 2016.

[6] A. Coates, B. Carpenter, C. Case, S. Satheesh, B. Suresh, T. Wang, D. J. Wu, and A. Y. Ng. large scale distributed deep networks. 2011.

[7] W. J. Dally. Route packet, not wires : On-chip interconnection networks. In *Proc. IEEE/ACM Design Automation Conference*, 2001.

[8] D. Das, S. Avancha, D. Mudigere, K. Vaidynathan, and P. Dubey. Distributed deep learning using synchronous stochastic gradient descent. 2016.

[9] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. 2018.

[10] D. Jia, D. Wei, R. Socher, L. J. Li, L. Kai, and F. F. Li. Imagenet: A large-scale hierarchical image database. *Proc of IEEE Computer Vision Pattern Recognition*, pages 248–255, 2009.

[11] N. P. Jouppi, C. Young, N. Patil, D. Patterson, and Gaurav Agrawal Et Al. In-datacenter performance analysis of a tensor processing unit. *Computer architecture news*, 45(2):1–12, 2017.

[12] A. Krizhevsky. One weird trick for parallelizing convolutional neural networks. *Eprint Arxiv*, 2014.

[13] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

[14] L. Li. Parallelized stochastic gradient descent. 2016.

[15] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, and B. Y. Su. Scaling distributed machine learning with the parameter server. *ACM*, 2014.

[16] E. Lindholm, M. J. Kligard, and H. Moreton. A user-programmable vertex engine. *ACM*, pages 149–158, 2001.

[17] S. A. Mojumder, M. S. Louis, Y. Sun, A. K. Ziabari, and A. Joshi. Profiling dnn workloads on a volta-based dgx-1 system. In *2018 IEEE International Symposium on Workload Characterization (IISWC)*, 2018.

[18] M. Naumov, D. Mudigere, Hjm Shi, J. Huang, N. Sundaraman, J. Park, X. Wang, U. Gupta, C. J. Wu, and A. G. Azzolini. Deep learning recommendation model for personalization and recommendation systems. 2019.

[19] P. Patarasuk and Y. Xin. Bandwidth optimal all-reduce algorithms for clusters of workstations. *Journal of Parallel Distributed Computing*, 69(2):117–124, 2009.

[20] P. Patarasuk and X. Yuan. Bandwidth efficient all-reduce operation on tree topologies. In *IEEE International Parallel Distributed Processing Symposium*, 2007.

[21] E. Qin, A. Samajdar, H. Kwon, V Nadella, and T. Krishna. Sigma: A sparse and irregular gemm accelerator with flexible interconnects for dnn training. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2020.

[22] S. Rashidi, P. Shurpali, S. Sridharan, N. Hassani, and T. Krishna. Scalable distributed training of recommendation models: An astra-sim + ns3 case-study with tcp/ip transport. In *2020 IEEE Symposium on High-Performance Interconnects (HOTI)*, 2020.

[23] S. Rashidi, S. Sridharan, S. Srinivasan, M. Denton, and T. Krishna. Efficient communication acceleration for next-gen scale-up deep learning training platforms. 2020.

[24] S. Rashidi, S. Sridharan, S. Srinivasan, and T. Krishna. Astra-sim: Enabling sw/hw co-design exploration for distributed dl training platforms. In *2020 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2020.

[25] G. F. Riley and T. R. Henderson. *The ns-3 Network Simulator*. Modeling and Tools for Network Simulation, 2010.

[26] Bernhard Schlkopf, John Platt, and Thomas Hofmann. Map-reduce for machine learning on multicore. In *Advances in Neural Information Processing Systems 19: Proceedings of the 2006 Conference*.

[27] F. Seide, H. Fu, J. Droppo, G. Li, and D. Yu. 1-bit stochastic gradient descent and its application to data-parallel distributed training of speech dnns. *interspeech*, 2014.

[28] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. Lecun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *Eprint Arxiv*, 2013.

[29] Chen Sun, Chia-Hsin Owen Chen, George Kurian, Lan Wei, Jason Miller, Anant Agarwal, Li-Shiuan Peh, and Vladimir Stojanovic. Dsent-a tool connecting emerging photonics with electronics for opto-electronic networks-on-chip modeling. In *2012 IEEE/ACM Sixth International Symposium on Networks-on-Chip*, pages 201–210. IEEE, 2012.

[30] D. Tarditi, S. Puri, and J. Oglesby. Accelerator: using data parallelism to program gpus for general-purpose uses. *ACM SIGOPS Operating Systems Review*, 40(5):325–335, 2006.

[31] R. Thakur, R. Rabenseifner, and W. Gropp. Optimization of collective communication operations in mpich. *International Journal of High Performance Computing Applications*, 2005.

[32] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, and K. Macherey. Google's neural machine translation system: Bridging the gap between human and machine translation. 2016.

[33] Y. You, I. Gitman, and B. Ginsburg. Scaling sgd batch size to 32k for imagenet training. 2017.

[34] MD Zeiler and R. Fergus. Visualizing and understanding convolutional neural networks. *European Conference on Computer Vision*, 2013.

[35] S. Zhang, A. Choromanska, and Y. Lecun. Deep learning with elastic averaging sgd. *MIT Press*, 2014.

[36] Shenyi Zhao and Wujun Li. Fast asynchronous parallel stochastic gradient descent: a lock-free approach with convergence guarantee. *AAAI Press*, 2016.

[37] S. Zheng, Q. Meng, T. Wang, W. Chen, N. Yu, Z. M. Ma, and T. Y. Liu. Asynchronous stochastic gradient descent with delay compensation. 2016.