ARIES: Accelerating Distributed Training in Chiplet-based Systems via Flexible Interconnects

Lingxiang Yin[1], Amir Ghazizadeh[1], Ahmed Louri[2], Hao Zheng[1]
[1]*Department of Electrical and Computer Engineering, University of Central Florida, Orlando, FL, USA*
[2]*Department of Electrical and Computer Engineering, George Washington University, Washington, D.C., USA*
[1]{lingxiang.yin, amir.g, hao.zheng}@ucf.edu, [2]louri@gwu.edu

***Abstract*—Large-scale deep learning models are widely deployed in many application domains with remarkable performance improvements. However, training these models with immense parameters calls for unprecedented computing and communication capabilities. Recently, chiplet-based architectures have shown much promise in scaling Deep Neural Network (DNN) inference, but their applications in the training phase remain unexplored and challenging.**

In this paper, we posit, beyond scaling computing capability, chiplet-based architectures could also be leveraged to enable new optimization opportunities for existing parallel training algorithms (e.g., Ring and Tree-based all-reduce). Specifically, we aim to explore a variety of topological characteristics, along with the interposer technology, to sustain the performance scaling of parallel training in chiplet-based systems. We propose ARIES, a versatile chiplet-based communication architecture supporting various parallel training algorithms using a flexible interconnect design. The proposed design can adapt to various collective operations such as reduce and gather across a wide diversity of training algorithms. Moreover, such flexibility is also leveraged to further enhance existing all-reduce algorithms depending on the latency and bandwidth requirements of the DNN model and dataset size. Simulation results show that the proposed ARIES can achieve up to 3.92× speedup in execution time and 38.8% reduction in Network-on-Chip (NoC) energy consumption when compared to prior work.

***Index Terms*—DNNs, parallel training, chiplets, collective operations**

I. Introduction

Deep learning is pervasive across numerous application domains such as computer vision, natural language processing, and speech recognition [1]–[4]. Despite the pervasiveness, the prominent success of deep learning relies on the continued increase of model size and complexity with billions of parameters [5], [6]. The training process for these complex and highly parameterized models is both time-consuming and costly, and thus it is posing unprecedented computational and communication challenges on the underlying hardware.

Chiplet-based technology has shown much promise in scaling deep learning inference with extended computing capability, but existing solutions [7], [8] have very limited applicability to the training phase. The primary barrier is the distinct communication characteristics presented in various parallel training algorithms. For example, Simba [7] exploited the benefits of chiplet architectures to sustain the performance scaling for deep neural network (DNN) inference in a cost-effective manner, but the inter-chiplet communication issue remains a challenge given the adoption of a traditional mesh topology. However, inter-chiplet communication is inevitable in a distributed DNN training system, as the gradients must be accumulated and exchanged among chiplets to achieve optimal convergence rate and model accuracy. In addition, SPACX [8] leverages the salient feature of silicon photonics to facilitate broadcast communications for DNN inference. Even though the broadcast capability can improve the spatial locality when performing DNN computations, it is unlikely to benefit distributed training where its gradient reduction (e.g., hop-by-hop data movement) is the primary concern.

On the other hand, current parallel training algorithms are primarily developed for distributed computing systems with compromised performance due to the agnostic nature and rigidity of their network topologies. For example, ring-based all-reduce and its variants are applicable to a wide range of topologies thanks to their simplicity. However, such simplicity comes with significant network resource under-utilization and long latency (i.e. gradient update), as this latter is bounded by the long ring diameter. Tree-based all-reduce provides a logarithmic reduction in network diameter, but, unfortunately, it could potentially suffer from network contention and bandwidth issue when deployed in a grid-like topology. Afterward, high-dimensional ring (e.g., 2D ring) and multi-tree (e.g., binary tree) are proposed to reduce the ring diameter and improve tree bandwidth. However, unlike off-chip networks, such high-dimensional properties are unable to be adequately managed on chips due to cost and power limits.

In this paper, we explore alternative topological characteristics to sustain the performance scaling of the ring and the tree-based all-reduce in chiplet-based systems from both interconnect and algorithm standpoints. The crux of our idea is to synergize the hardware and algorithm designs to leverage the characteristics of hierarchical and concentrated topology, rather than high dimensionality, to maintain the performance scaling of all-reduce in chiplet-based systems. Specifically, we propose **ARIES**, an efficient communication framework, along with improved all-reduce algorithms, for parallel training in chiplet-based systems. We identify that network resource utilization could be a critical factor to reflect the communication performance of all-reduce collectives in addition to generic metrics (e.g., diameter and bi-section bandwidth). Furthermore, a flexible interconnect design is proposed to enable various means of hierarchical and concentrated optimizations increasing the network utilization in ring and tree-structured all-reduce. The proposed all-reduce algorithms are thus tailored to align with on-chip topological characteristics to improve overall throughput and reduce synchronization latency.

The specific contributions of this paper are listed as follows:

- **ARIES Architecture:** We propose a chiplet architecture that supports a wide range of parallel training algorithms. The interconnect of the proposed ARIES architecture can decouple the stiff connectivity between router ports and links. As such, intra-router connectivity can be dynamically changed, maximizing network resource utilization. In addition, a multi-function link design is proposed to support both dynamic link segmentation and connectivity, increasing overall link count and avoiding traffic contention.
- **ARIES All-reduce Algorithms:** We revisit the major issues of deploying all-reduce algorithms in conventional grid-like on-chip topologies. As compared to off-chip counterparts, the proposed all-reduce algorithms are optimized for on-chip networks with low dimensions and are independent of the rigid and agnostic network topology.
- **Detailed Evaluation:** We evaluate the proposed ARIES using a cycle-accurate system simulation [9] with ResNet-50 [10], VGG-16 [11], DLRM [12] and Transformer [13] benchmark

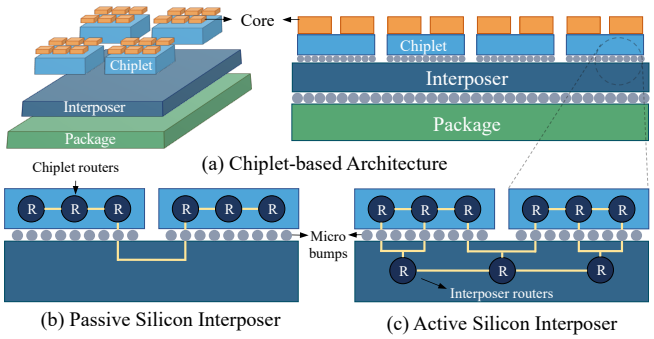


Fig. 1: (a) Chiplet-based architecture, (b) passive interposer integration, and (c) active interposer integration.

suites. Our simulation results show that ARIES can achieve up to 3.92× speedup in execution time and 38.8% reduction in Network-on-Chip (NoC) energy consumption when compared to prior work [14], [15].

II. Background and Motivation

A. Chiplet-based Architectures

Advanced packaging technology has emerged to disintegrate a large chip into multiple small chips, called chiplets, to increase yield and reduce manufacturing costs. Chiplets are combined and interconnected to compose a larger and more complex system. Figure 1, shows the cross-section view of a chiplet-based design. In this vertical stack-up, a silicon interposer is mounted on top of the package to accommodate inter-chiplet communications. Additionally, the chiplets are placed on an active or passive interposer. As shown in Figure 1(a), passive interposer designs are simpler and primarily serve as a routing layer, while active interposer designs contain active components that can provide additional logic functionality, such as signal processing or power management. The logical functionalities could be utilized to speed up inter-chiplet communications. The chiplet-based architectures are being used for many commercial products [16], [17] and academic prototypes. However, to the best of our knowledge, there are relatively few works to date that have explored the chiplet-system design to support DNN training.

B. Distributed Deep Neural Network Training

Large-scale DNNs models necessitate substantial computational resources for their training. To mitigate the computational burden and complete the training process within an acceptable time frame, numerous research has focused on distributed training techniques aiming to accelerate the learning process [18]. The common approach employed for training DNNs is data parallelism, which refers to dispatching a model replica to each node, where a separate batch of data is processed locally in parallel. [19] Concretely, each node will produce a unique set of Local Gradients (LGs) w.r.t the loss function that must be collected from all the nodes and then reduced (e.g., summed or averaged). Once the reduction is performed, all the nodes share the same parameters referred to as Global Gradients (GGs), where the distributed system can optimize the parameters using Stochastic Gradient Descent (SGD) [20]. In this paper, we focus on data parallelism distributed training systems.

A distributed training task involves three steps regardless of the DNN variants. (1) The forward pass to calculate the loss function, (2) computing the gradients of the parameters w.r.t the loss using backpropagation, and (3) optimizing the parameters using the SGD

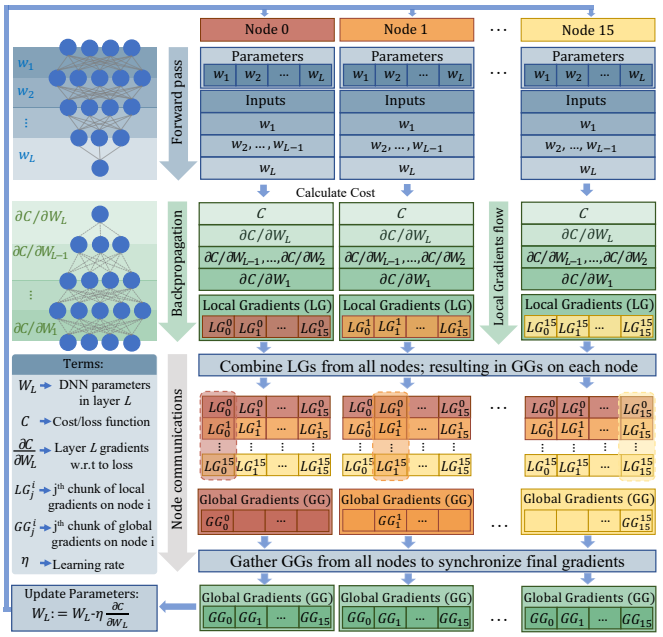


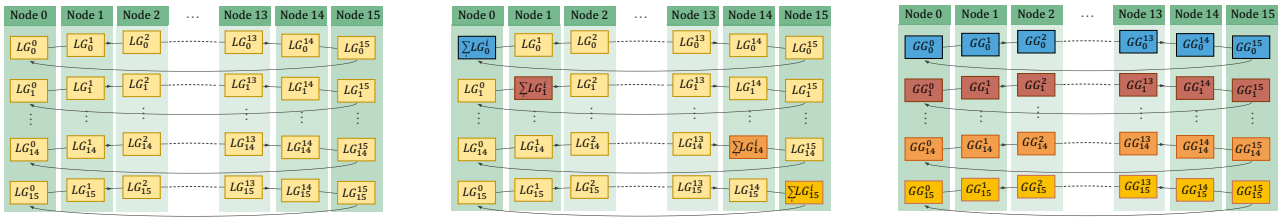
Fig. 2: An overview of DNN distributed training on 16 nodes.

algorithm. The first step is a simple forward pass and is not quite challenging. In the second step, however, the obtained gradients after backpropagation are local to each node that must be accumulated, because optimizing the model parameters requires all nodes to synchronize their gradients uniformly.

Figure 2 illustrates an example of a DNN distributed training using 16 nodes. Using the same model replica ($W_1, ..., W_L$), each node takes charge of processing one data chunk and producing one gradient chunk (i.e., LG_j^i) during the forward and backward pass, respectively. Consequently, all nodes must communicate to send/receive their gradient chunks. Since the gradient tensor has a large volume, especially in large-scale DNNs, each node sends/receives a slice of the gradient tensor at a time for the gradients to be combined (e.g., averaged). The local gradients on each node are sliced into 16 parts. For synchronization, each node is responsible for combining one of the 16 slices based on its index, meaning node i is in charge of combining the ith gradient chunk, resulting in a global gradient for that slice (GG_i). Next, the nodes must communicate one more time to share the final global gradients and update the model based on SGD. This process occurs for every single iteration of training. Having the global gradients as a whole, the model can update its parameters using SGD and be ready to go through the next training iteration. The challenge is that the training time is bounded by the extensive communication caused by gradient synchronization [21].

C. Collective Communication Algorithms

In distributed training, the calculated gradients must be synchronized among computing nodes before each weight update step. Two common approaches are considered to synchronize the gradients, namely centralized and decentralized [22]. In the centralized approach, all the gradients will be accumulated at a given node, posing high bandwidth requirements to the endpoint. In contrast, the decentralized approach performs gradient exchange among computing nodes via all-reduce operations [23] shown in Figure 2. The all-reduce operation performs reductions (e.g., sum, max or average) on the local gradients across nodes.



(a) Initial data on each node (b) Reduce-Scatter (c) All-Gather

Fig. 3: Ring-based all-reduce (reduce-scatter and all-gather) operations: (a) data distribution (input, weight, and gradient) in an accelerator with 16 nodes, (b) Ring-based reduce-scatter operation to accumulate gradient, and (c) Ring-based all-gather operation to update weight.

TABLE I: Bandwidth and Latency Analysis of All-reduce Algorithms in Torus

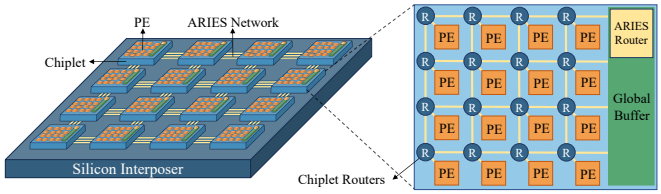
All-reduce	Max # of Links	Bandwidth/Link	Synchronization Latency	Max Bandwidth
Ring	N	G/N	$N-1$	G
2D Ring	2N	G/N	$2(\sqrt{N}-1)$	$2G$
Binary Tree	$2*(N-1)$	$G/2$	$\sqrt{N}-1$	$\approx$ $2G$
Multi-Tree	$4*(N-1)$	$G/4$	$\sqrt{N}-1$	$\approx$ $4G$
ARIES	$8(\sqrt{N}-1)*\sqrt{N}$	G/N	$\min(\log_4 N/4,\ L\times(\sqrt[L]{N}-1))$	$4G$

Among many all-reduce algorithms, the ring all-reduce algorithm has been integrated into commercial collective libraries [24], [25]. In this algorithm, the all-reduce operation consists of two steps, reduce-scatter and all-gather, which corresponds to the combine and gather phase of Figure 2, respectively. Figure 3 illustrates the stepwise process to perform all-reduce in a 16-node distributed system.

We take node 0 as an example to illustrate the reduce-scatter operation (the first row in Figure 3(b)). At time step 1, node 1 will send its gradient (LG_0^1) to node 2. In the next step, node 2 will receive the gradient (LG_0^1) from node 1 and adds it to LG_0^2. Following this principle, the gradient LG_0^i will be accumulated with all the gradients stored at intermediate nodes all the way to the end node 0. Similarly, each node has a part of the accumulated gradients ($\sum_i LG_0^i$) by time step 16. Upon the completion of the reduce-scatter, an all-gather operation as shown in Figure 3(c), is performed. Each node will later broadcast the accumulated gradients ($GG_i = \sum_i LG_0^i$) to the rest of the nodes hop by hop. Finally, all the accumulated gradients will be received at each node.

D. Motivation

Current all-reduce algorithms are built upon ring and tree structures, providing either linear or logarithmic scaling in network bandwidth and synchronization latency. Theoretically, these algorithms [18], [26], [27] leverage topological characteristics, such as high dimensionality, to improve their performance in synchronization latency and bandwidth. In the meantime, this necessitates high-dimensional topologies, such as hypercube, to meet the bandwidth and latency requirements. Given the power and area constraints in chiplet-based systems, the performance of existing all-reduce algorithms could be bounded by the limited and rigid network connectivity. As shown in Table I, we formulate the bandwidth and latency performance of various all-reduce in a $\sqrt{N}\times\sqrt{N}$ torus network. For example, even though ring all-reduce can reduce the link bandwidth requirement to G/N, in which G is the gradient size for the entire model. However, the synchronization latency is bounded by the ring diameter ($N-1$), as the gradients need to be accumulated across the network. To resolve this problem, the ring diameter can be reduced radically ($2\sqrt{N}-1$) by having high-dimensional rings such as 2D ring, but the increased dimensionality requires doubled link count. On the other hand, we assume K as the tree number and D as the degree for each node. The tree-based all-reduce can enable logarithmic scaling in synchronization latency, but it has a relatively high bandwidth requirement per link. The bandwidth per link is proportional to the tree count (i.e. K), and, similarly, the reduced requirement comes with overheads in link count ($K*(N-1)$). Eventually, the performance of all-reduce algorithms on chips could be bounded by the effective utilization of wiring resources ($4N$ in torus) and network diameter ($\sqrt{N}-1$). With the limited area and wiring budget, it requires an efficient algorithm and hardware co-design to better utilize network resources facilitating all-reduce collectives. Our proposed design, ARIES, aims to provide alternative latency reduction and obtain maximum bandwidth utilization in chiplet-based architectures while retaining similar area and wire costs as compared to the torus. Please note that L is the depth of hierarchical topologies.



(a) (b)

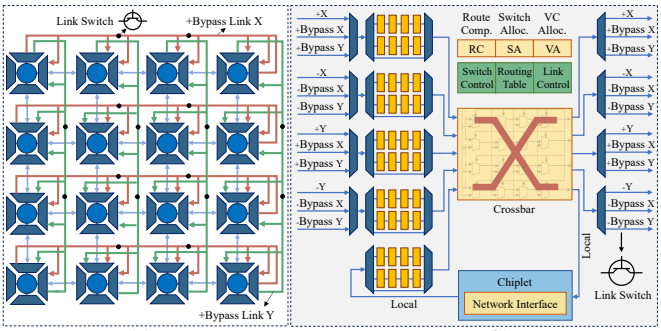
Fig. 4: ARIES architecture in (a) package-level, and (b) chiplet-level.

III. Proposed ARIES Design

The primary goal of ARIES is to facilitate prevalent collective communications that arise in distributed training. In particular, ARIES consists of two salient designs: a flexible interconnect design and improved all-reduce algorithms. The flexible interconnect can increase link count and rematch the router port and link, which can effectively increase network utilization with ultra-low overheads. The proposed all-reduce algorithms pursue alternative topological characteristics and flexibility to sustain the performance scaling of collective communications. The proposed hardware and algorithm co-design can remedy the compromised performance caused by the misalignment between network topology and all-reduce algorithms.

A. ARIES Micro-architecture

Figure 4 shows ARIES architecture at two levels - package and chiplet. At the package level, as shown in Figure 4(a), ARIES consists of an array of N×N chiplets interconnected by a flexible interconnect, which we call ARIES NoC in this paper. Each chiplet has an array of M×M TPU-like PEs and a global buffer connected by a mesh interconnect, as shown in Figure 4(b). For simplicity, we use a 4×4 ARIES architecture to illustrate the concept.



(a) (b)

Fig. 5: ARIES architecture in (a) a 4×4 ARIES package-level NoC architecture, and (b) proposed ARIES router architecture.

B. ARIES NoC Architecture

Unlike off-chip networks, high-dimensional topologies are hardly being deployed on chips to address the performance bottlenecks of collective communications due to the limited and rigid metal layers. As such, improving network utilization and flexibility could be an alternative solution to sustain performance scaling concerns. Even though prior work [28]–[31] has studied flexible interconnect designs, they mainly focused on altering the inter-router connectivity, thereby reducing long-haul communication and increasing bi-section bandwidth. In all-reduce operations, resource underutilization is mostly caused by insufficient intra-router connectivity. Moreover, the link resources, regardless of their length and connectivity, are not enough to support hop-by-hop gradient accumulations. Unfortunately, those issues are not well addressed in the existing literature.

For example, mapping a binary-tree all-reduce algorithm [14] to a $\sqrt{N}\times\sqrt{N}$ grid-like topology could lead to resource under-utilization. The binary tree all-reduce algorithm requires $2(N-1)$ connections between nodes, whereas the torus topology can provide $4N$ links. This indicates that a significant amount of wires and routers are underutilized, not to mention ring-based algorithms (with fewer link requirements). On the other hand, K-tree all-reduce requires $K \times N$ edges, which overwhelms the overall wire budget when K is larger than four.

To this end, the proposed ARIES NoC design adds an additional degree of flexibility to conventional router architecture, enabling improved resource utilization for all-reduce algorithms. Additionally, we proposed a low-cost reconfigurable link that can be sized to a bundle of short wires increasing total link resources. For simplicity, we utilize a 4×4 ARIES NoC to illustrate the key idea. As shown in Figure 5, ARIES NoC adopts a passive interposer design, where only wires are deployed in the silicon interposer. The router, including its control logic, is implemented on the chiplet. ARIES NoC is built on top of a mesh topology, where two bi-directional bypassing links are placed across each row and column. ARIES NoC can adapt to various network topologies supporting any tree and ring-based all-reduce algorithms.

1) ARIES Router: As mentioned, a large number of router ports and links are underutilized in ring and tree all-reduce algorithms. The root cause is the restricted connectivity between the router port and network link, as each router port is only connected to a given link in conventional NoC design. To resolve this issue, we can either connect one port with multiple links to relax this constraint or enable link selection at each port with multiple possible connections. In addition, to address the bandwidth issue caused by the micro-bump pitch size, we still follow a virtual channel design in the router, which avoids head-of-line blocking and increases throughput.

ARIES Router is similar to the conventional mesh-based router with five input ports, +x, -x, +y, -y, and local. In addition to the conventional router, a set of muxes are added into +x, -x, +y, and -y directions, and each mux connects to a mesh link, and two uni-directional bypassing links are placed at x and y direction. For example, the input port +x is connected to +x, +bypassing x, and +bypassing y. This allows the input port +x can be utilized for receiving packets from three links +x, +bypassing x, and +bypassing y. Please note that the router radix does not increase, and only one link can be connected to one port at a time. At the virtual channel of each input port, a link is implemented to support circuit switching, where all the data can be directly injected into the local port without buffering at the router. The routing table records the ring and tree connectivity, and it records the upstream and downstream router information. The routing information can be configured when the specific all-reduce algorithm is determined. We use round-robin arbitration to ensure the fairness of packets.

2) ARIES Link: A variety of links have been discussed in [26], and the key design philosophy is to bridge long-distance communication. However, those long-haul links, like wrap-up links in the torus, are underutilized in the ring all-reduce algorithm, as its communication only happens between adjacent nodes. On the other hand, the tree topology desires long-distance communication to avoid contention, thereby maximizing its advantages in latency reduction. It is costly to accommodate both requirements when considering the limited wiring budget. To address this issue, we propose the ARIES link, and its idea is very similar to an adaptable link design but with a much lower cost. As shown in Figure 5, a simple transistor is implemented in each bypassing link. The transistor acts like a switch to turn on/off the wire connection with its associated router. This allows the desired sizing of links.

IV. All-reduce Algorithms in ARIES

Collective communication primitives have been proposed to speed up parameter synchronization in distributed learning, such as NVIDIA's Collective Communications Library (NCCL), Uber's Horovod, and Baidu's Ring all-reduce. Despite the aforementioned, the direct application of all-reduce algorithms in on-chip networks remains unexplored, requiring a careful study of various all-reduce algorithms. In this section, we argue that the algorithm and communication fabrics should be synergized toward parallel distributed training.

A. Ring-based All-reduce Optimization

The ring all-reduce algorithm and its variants have been widely implemented to support distributed learning in existing commercial libraries. The advantage of ring-based algorithms is the bandwidth, where gradients are partitioned and accumulated in a distributed manner. This requires each set of gradients to traverse the entire network to complete the Reduce operation, as shown in Figure 6. For example, a chunk of the gradients is calculated at node 1, which will be accumulated with the gradients stored in other nodes later. As such, the latency of completing the ring-based Allreduce operation is limited by the topology diameter. In addition, mapping a ring-based algorithm in a grid-like topology is likely to result in a resource under-utilization issue, where a large set of links and router ports are not fully utilized, as shown in Figure 6(a). The key idea of ARIES design is to utilize idle network resources to improve the performance of ring-based all-reduce algorithms. The idle resources

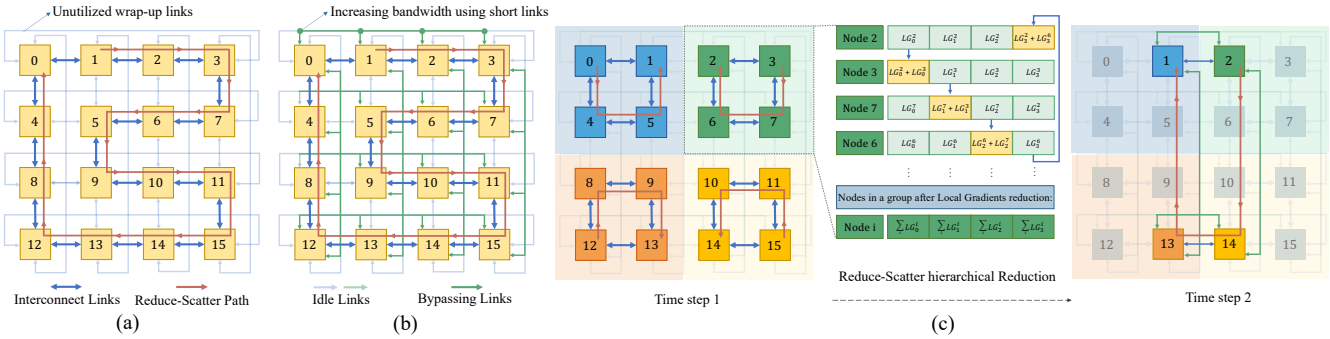


Fig. 6: Optimized Ring-based all-reduce in ARIES: (a) mapping of ring all-reduce in a 4×4 torus, (b) bandwidth optimized ring all-reduce in a 4×4 ARIES, and (c) latency optimized ring all-reduce in a 4×4 ARIES.

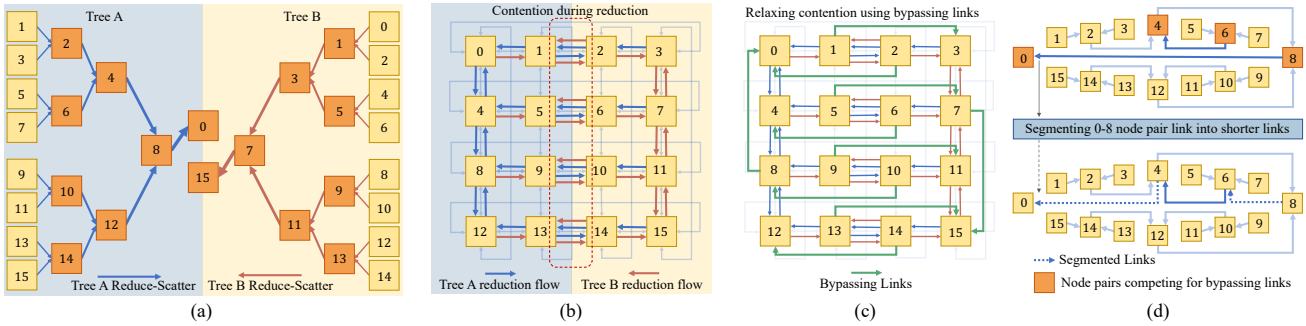


Fig. 7: Optimized Double Binary Tree in ARIES: (a) double binary tree all-reduce algorithm (Tree A and Tree B), (b) traffic contention of double binary tree in a 4×4 Torus, (c) mapping of double binary tree in a 4×4 ARIES, and (d) tree scalability issue.

could be utilized to either reduce the network diameter or double the network bandwidth. However, this requires adequate modifications in the network and algorithm. We illustrate two approaches to optimize the bandwidth and latency performance of ring-based all-reduce algorithms.

1) Bandwidth Optimized Ring All-reduce in ARIES: Bandwidth optimization utilizes idle network resources to double the bandwidth between adjacent routers. The essence of this technique is to increase the number of links that could possibly connect adjacent routers, and thus $2N$ links are used. In ARIES, the long-haul bypassing links should be segmented into a number of short links, thus providing additional connections between adjacent routers. However, the input ports have been occupied by the mesh links. For example, as shown in Figure 6(b), for routers 9 and 10, the segmented bypassing links at x direction (green color) are unable to be connected to the input ports that have been used by mesh links (blue color). In such a case, those segmented bypassing links can be connected to idle router ports in the y direction. Since the bandwidth optimization only requires the change at the NoC connectivity, the ring all-reduce algorithm remains the same communication pattern.

2) Latency Optimized Ring Topology in ARIES: As mentioned, the network diameter determines the latency to complete the gradient update in the ring all-reduce algorithm. Given the fact that gradients have to be accumulated node by node, conventional bypassing techniques are no longer useful. Based on this observation, we propose a hierarchical ring-based all-reduce that can minimize the latency of gradient synchronization. The key idea is that multiple ring all-reduce operations are performed simultaneously to reduce the number of hops required to complete the synchronization, thereby multiple rings will be formed in support of the all-reduce operations at the same time. Partially accumulated gradients will be further accumulated as a ring. The hierarchical design can effectively reduce the latency.

In conventional grid-like topology, the ring all-reduce takes 16 steps to complete the reduce-scatter. In the proposed hierarchical ring topology, all nodes within each quadrant (denoted as four different colors) will perform reduce-scatter operations in parallel. As shown in Figure 6(c), the gradients will be accumulated from node 3 to node 2 in a clockwise direction at the upper right quadrant. As such, the gradients at four quadrants will be accumulated at nodes 1, 2, 13, and 14.

Following that, the accumulated gradients from the four quadrants will perform a clockwise or anti-clockwise accumulation. Consequently, the latency of completing the reduce-scatter operation is reduced by half as compared to the traditional ring all-reduce at the cost of a higher bandwidth requirement.

This methodology can be generalized to a $\sqrt{N} \times \sqrt{N}$ network, in which a L-level hierarchical ring all-reduce is configured. As such, the gradient accumulation will be performed hierarchically. The overall latency can be reduced to $L \times (\sqrt[L]{N} - 1)$. In such a case, multiple $\sqrt[L]{N} \times \sqrt[L]{N}$ rings will be formed to perform partial gradient accumulations in parallel.

B. Tree-based Collective Optimization

Even though tree-based all-reduce algorithms can significantly reduce the latency due to the logarithmic reduction of tree height, the deployment of such algorithms to on-chip networks faces two major issues. First, given the grid-like topology, the physical distance between tree nodes varies, leading to traffic contention. Second, wire resources and micro-bump pitch are not sufficient enough to support all the link connections between tree nodes.

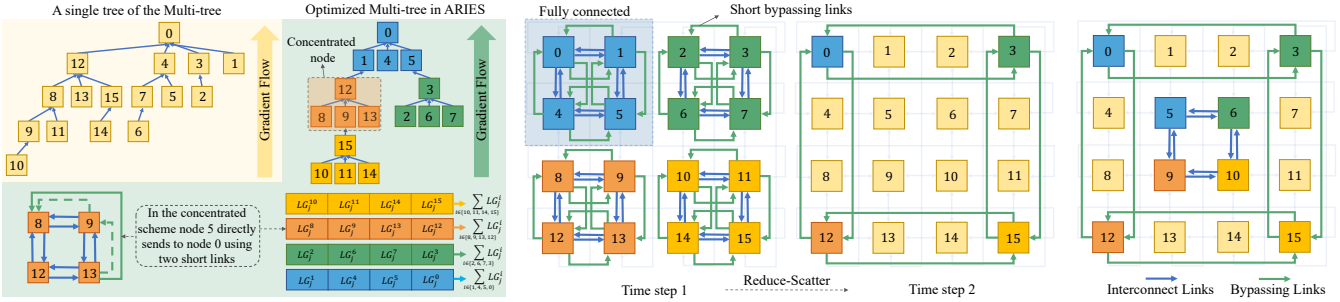


Fig. 8: Optimized Multi-tree in ARIES: (a) Concentrated Multi-tree all-reduce in ARIES, (b) mapping of concentrated multi-tree in ARIES, and (c) bandwidth optimization in multi-tree all-reduce.

For any tree-based algorithms [32]–[34], we assume K trees are needed to perform all-reduce operations in a $\sqrt{N} \times \sqrt{N}$ network. Considering the limited network resources, there is a trade-off between the number of trees and the bandwidth requirement. Building more trees can alleviate the bandwidth requirement for gradient propagation. The latency of each tree is determined by the tree height ($log_D N$), in which the logarithm base (D), the number of children of each node, is the critical factor. As such, the total number of edges of K trees is $K \times N$, and each edge demands an NoC link in an ideal case. For a torus topology, the maximum number of links is $4N$ (bi-directional links are counted). It implies that NoC links may not be enough to support each tree edge when K is larger than four, regardless of wire length. Given this, we propose two optimization techniques to manage the mentioned two scenarios. For simplicity, we select the two most commonly seen tree-based all-reduce algorithms, double binary tree and multi-tree, as case studies.

1) Binary Tree Optimization in ARIES: Double binary tree all-reduce [32], [33] is another algorithm for collective operations and is well supported in commercial libraries [25]. As shown in Figure 7(a), two binary trees are constructed to complete the all-reduce operations simultaneously. Theoretically, the double binary tree has $2N$ edges, and it is within the wire budget. The primary issue is the mismatch between the tree graph and the physical layout - leading to long-distance communication and traffic contention, as shown in Figure 7(b). For example, nodes 10 and 12 are adjacent nodes in the tree graph but are not physically connected. The physical distance between the tree nodes could increase when the network size scales. This requires the application of long-haul links to bridge non-adjacent nodes.

Figure 7(c) illustrates an example of configuring ARIES NoC to build a dedicated connection for the double binary trees (Tree A and B). Each link is allocated for a given route in the tree. For example, nodes 1 and 3 in Tree B are connected via the bypassing link. As such, the long-haul bypassing links can bridge non-adjacent nodes in the network, reducing network latency and avoiding contention. The hop count reduction for each tree layer could be generalized as Equation 1, where h is the height level, and $\sqrt{N}$ is the network dimension size.

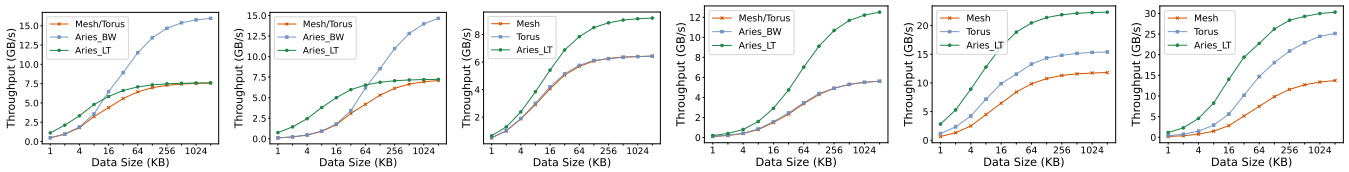
$$Hop\ Count\ Reduction_h = \begin{cases} \sqrt{N}/2^h - 1, & 2^h \leq \sqrt{N} \\ N/2^h - 1, & \text{otherwise} \end{cases} \quad (1)$$

Even though both binary trees (Tree A and B) are well-fitted into ARIES NoC, the bypassing links are not enough for each row or column when the network size is larger than 16×16. For example, in Figure 7(d), two pairs of nodes (nodes 0-8 and 4-6) compete for the bypassing links. In such a case, we divide the bypassing links into multiple segments to satisfy the shorter bypassing links. For a larger network size, we follow a greedy policy to segment the bypassing links to connect any short-distanced nodes. The primary reason behind this design is to simplify the routing algorithm to prevent any channel dependencies.

2) Multi-tree Optimization in ARIES: In a distributed training system consisting of n nodes based on the Multi-tree algorithm [34], Figure 8(a) illustrates a single tree from the n constructed trees. Each one of these trees is solely in charge of sending/receiving a slice of the gradient tensor by index, such that the first constructed tree performs the reduction of the first gradient chunk, i.e., $\sum_i LG_0^i$, while the jth tree is reducing the gradient chunk j, i.e., $\sum_i LG_j^i$ (note that i is the node index). Once all the n trees complete the reduction for the entire chunks of the gradients, each tree has the global gradient based on its index (GG_i). Finally, the all-gather operation is performed to synchronize the global gradients.

Compared to the double binary tree algorithm, the multi-tree algorithm [15] attempts to fully utilize all the network links by having short-distance communications. However, even though the network bandwidth is fully utilized, the bandwidth of multi-tree is bounded by the link count, and its tree height still limits the synchronization latency. In other words, the tree's height is bounded by the diameter of the physical topology layout. It performs even worse when the traffic of multiple trees saturates a given node.

To tackle this problem, our key idea is to reduce the diameter of the topology. Generally, two design choices can be used to reduce the network diameter in the mesh-based topologies - adding extra links (bypassing) or increasing router radix (concentration). We propose a tree concentration approach to group a set of nodes as a concentrated node as illustrated in Figure 8(a). The bypassing links can be segmented into multiple short links, providing direct connections for four nodes. As shown in Figure 8(b), in the blue concentrated node, all the gradients (LG_j^0, LG_j^1, LG_j^4, and LG_j^5) will be accumulated at a single node ($\sum_i LG_j^i$, where $i \in \{0, 1, 4, 5\}$). Once the data is reduced or gathered, concentrated nodes will perform normal multi-tree operations as shown in Figure 8(c). However, it could increase the bandwidth requirement on the link, as the concentration reduces the number of trees. To address this issue, we can partition the gradient accumulation into two nodes within each concentrated tree node. As such, we fully utilize all the links at each step while reducing the tree height by half as shown in Figure 8(c). As a result, the latency of the proposed optimization is approximately reduced to $log_D N/4$, and it attains a similar level of bandwidth as a multi-tree.



(a) 4×4 Ring all-reduce (b) 8×8 Ring all-reduce (c) 4×4 DB-T all-reduce (d) 8×8 DB-T all-reduce (e) 4×4 MT all-reduce (f) 8×8 MT all-reduce

Fig. 9: Throughput Analysis of ring and tree all-reduce Algorithms on different network topologies with various data size: (a) Ring all-reduce in 4×4 Networks, (b) Ring all-reduce in 8×8 Networks, (c) Double Binary Tree all-reduce in 4×4 Networks, (d) Double Binary Tree all-reduce in 8×8 Networks, (e) Multi-tree all-reduce in 4×4 Networks, and (f) Multi-tree all-reduce in 8×8 Networks.

TABLE II: System Configuration.

PE Parameters	Configuration	Accelerator Parameters	Configuration
Data Width	32 bits	# of PEs	16
Buffer Size	64 KB	Global Buffer	4 MB
Multiplier Array	32 × 32	Clock Rate	1 GHz

C. Deadlock Avoidance

The network deadlock occurs due to improper NoC reconfiguration, protocol deadlock, or circular channel dependence. In our design, NoC is only configured prior to running the application, which avoids any potential deadlocks caused by having distinct routing algorithms. Furthermore, we follow the standard approach, the deployment of virtual channels, to avoid protocol deadlocks. Circular channel dependence is caused by the routing algorithms. In this paper, the communication patterns of all the all-reduce algorithms naturally prevent forming circular channel dependence. Consequently, deadlock is intrinsically avoided in the proposed ARIES NoC design.

V. Evaluation

For our evaluation, we extended the open-sourced ASTRA-SIM simulator [9] to support a wide range of ring and tree-based collective algorithms. The network performance is evaluated through the GARNET simulator integrated with ASTRA-SIM. The detailed simulation setup is included in Table II. To evaluate the area and power consumption, we complete the synthesis for package, chiplet, and PE, including the interconnect, MAC Array, buffers, and DRAM. We use the Synopsys Design Compiler with the PDK $32nm$ library for the synthesis and estimate the power using Synopsys PrimeTime PX. Several DNN benchmarks are used to evaluate our proposed design, including ResNet-50 [10], VGG-16 [11], DLRM [12] and Transformer [13]. The baseline mesh router has two pipeline stages (look-ahead routing and speculative optimizations), four virtual channels, and 32 buffers per VC. We assume the inter-chiplet hop latency is 10 ns and the bandwidth is 68 GB/s.

A. Throughput Analysis for varying data sizes

As the proposed ARIES NoC design can adapt to various all-reduce algorithms, we evaluate the throughput of ARIES design on different network topologies with different data sizes (1 KB to 2 MB). Given the on-chip layout constraints, we mostly compare our proposed design to on-chip topologies such as mesh and torus.

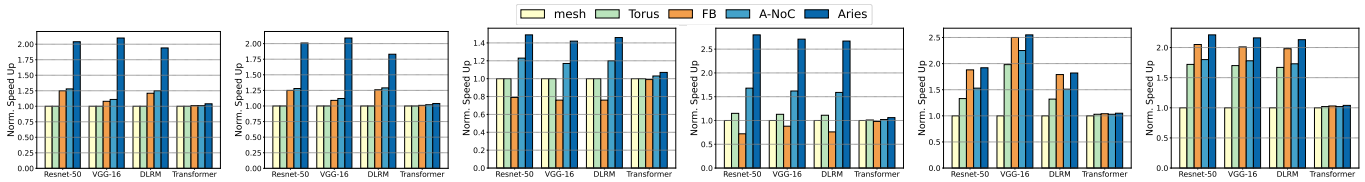
As shown in Figure 9(a) and (b), we compare two ARIES optimizations to target latency and bandwidth separately, which are denoted as ARIES_LT and ARIES_BW in the ring all-reduce. We observed that ARIES_LT performs much better than other topologies, including ARIES_BW when the data size is small. As the data size increases, the benefit of ARIES_BW is more significant. When the network size increases from 4×4 to 8×8, the throughput gain for ARIES_BW surpasses others with larger data sizes. It should be noted that the message size decreases when the node number increases.

We also studied the throughput of ARIES in two tree-based all-reduce algorithms, namely double binary tree (DB-T) and Multi-tree (MT). From Figure 9(c) and (d), we can infer that ARIES has a very similar throughput as compared to the mesh and the torus with small data sizes (1-8 KB), but the throughput gain climbs with the growth of data size. Similarly, when the network size increases to 64 nodes, the throughput of ARIES increases significantly regardless of the data size. This is because ARIES reduces more hops by adding bypassing links in a larger network. For the MT algorithm, as shown in Figure 9(e) and (f), we observed a similar trend in throughput; ARIES achieves high throughput improvements as compared to the mesh and the torus in a 16-node system, whereas the throughput increase is more significant in a larger network with 64 nodes. This phenomenon is caused by the optimized MT in ARIES, which utilizes the concept of concentration to reduce tree height, where the tree height decreases more significantly in larger networks.

B. Performance Analysis

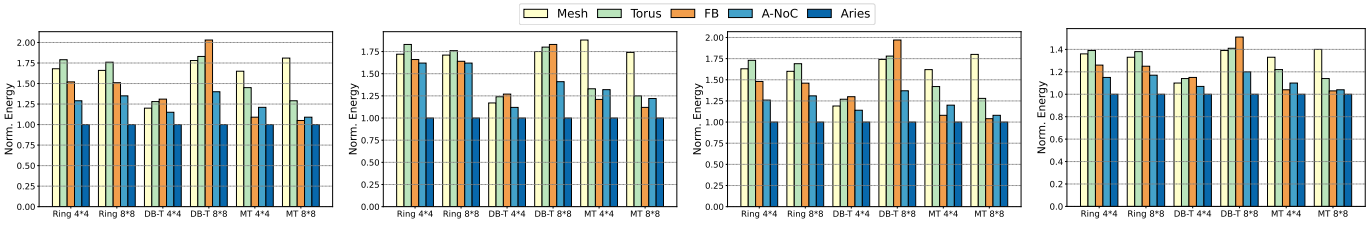
We further examine the performance (execution time) benefits of having ARIES NoC design with real-world benchmarks. We compare ARIES NoC with mesh, torus, Flattened Butterfly (FB) [35] and Adapt-Noc [28] topologies. It should be noted that we designed a pipeline to overlap the latency between communication and computation for all designs, as it is a common approach to optimize DNN training performance. For the ring all-reduce in Figure 10(a) and (b), the performance in the mesh and the torus topologies is almost identical, as it follows the same communication pattern. We observed that ARIES could achieve 2.10× speedup in a 4×4 network as compared to the mesh and the torus topologies, whereas it achieves 2.09× speedup in an 8×8 network. While FB and Adapt-Noc perform better compared to mesh and torus topologies, as multiple nodes are fully connected, which can reduce overall hops. ARIES still remains the best choice and achieves 1.52× and 1.48× speedup in a 4×4 network, and it has 1.46× and 1.42× speedup in an 8×8 network when compared to FB and Adapt-NoC, respectively. That is because, with larger data size, ARIES has a better performance in utilizing network resources for bandwidth optimization. For DB-T all-reduce algorithms, the performance of the mesh and the torus topologies is also similar, as the wrap-up links of the torus are not utilized for optimizing DB-T algorithms.

As Figure 10(c) and (d) show that ARIES can achieve 1.35×, 1.35×, 1.65×, 1.17× speedup in a 4×4 network as compared to mesh, torus, FB and Apadt-NoC topologies. Adapt-NoC performs relatively better, as it can adapt itself to fit the DB-Tree structure to some extent. FB performs worse in DB-Tree, as the multiple tree nodes compete for the same channel due to the high radix. The speedup of ARIES follows a similar trend in a larger network with 64



(a) 4×4 Ring all-reduce (b) 8×8 Ring all-reduce (c) 4×4 DB-T all-reduce (d) 8×8 DB-T all-reduce (e) 4×4 MT all-reduce (f) 8×8 MT all-reduce

Fig. 10: Performance Analysis of ring and tree all-reduce Algorithms on different network topologies with various DNN applications: (a) Ring all-reduce in 4×4 Networks, (b) Ring all-reduce in 8×8 Networks, (c) Double Binary Tree all-reduce in 4×4 Networks, (d) Double Binary Tree Allreduce in 8×8 Networks, (e) Multi-tree all-reduce in 4×4 Networks, and (f) Multi-tree all-reduce in 8×8 Networks.



(a) ResNet-50 (b) VGG-16 (c) DLRM (d) Transformer

Fig. 11: Energy Analysis of ring and tree all-reduce Algorithms on different network topologies with various DNN applications.

nodes, where its speedup increases up to 2.80×, 2.43×, 3.92× and 1.67× compared to other topologies, respectively. For MT all-reduce in Figure 10(e) and (f), ARIES can achieve 2.55×, 1.44×, 1.03× and 1.25× speedup in a 4×4 network as compared to mesh, torus, FB and Adapt-NoC topologies, and 2.21×, 1.28×, 1.08× and 1.23× speedup in an 8×8 network, respectively. FB performs better on MT, as it has a higher network radix, which helps to reduce MT's height. However, the rigid connectivity of FB still cannot outperform ARIES. Please note that in transformers training the overall speedup in execution time is relatively small. This is because the computation time for the transformers dominates the total execution time, and thus the overall speedup is not significant even though the communication time is reduced by 33.7% for ring all-reduce, 51.9% for DB-T all-reduce and 19.5% for MT Allreduce on average across different networks.

C. Energy Analysis

In this section, we evaluate the energy benefits of having ARIES with various DNN architectures. As shown in Figure 11, ARIES achieves 30.5% and 38.8% energy reduction on average in a 4×4 and an 8×8 mesh network with different DNN architectures and all-reduce algorithms, respectively. When compared to the torus, ARIES achieves 28.9% and 33.8% reduction on average in a 4×4 network and an 8×8 network. As compared to FB, the average energy reduction is 21.2% and 29.2% in a 4×4 network and an 8×8 network. The average energy reduction indicates a 17.5% and 20.7% improvement in a 4×4 network and an 8×8 network when compared to Adapt-NoC. Thanks to the adaptive bypassing links, the energy reduction stems from the total execution time and the overall hop counts being reduced. MT all-reduce benefits the most, as it achieves the highest speedup and most hops reduction when compared to other algorithms.

D. Area Overhead

We evaluate the area overhead through Synopsis Design Vision using 45 nm and 65 nm technologies for the chiplet and the interposer, respectively. The baseline chiplet router consists of a crossbar, a switch allocator, a virtual channel allocator, and buffers of size 17806 um^2, 4589 um^2, 9066 um^2, 98740 um^2, respectively. As a result, the overall chiplet NoCs account for 8.3 mm^2 area. The ARIES requires 0.3 mm^2 additional area on the chiplet as compared to the mesh topology.

VI. Related Work

Applications have been observed to exhibit varying behaviors and characteristics over time [36]–[38], necessitating an adaptable on-chip interconnect to meet their distinctive communication demands. To this end, Runtime reconfiguration has been proposed to improve NoC performance, reliability, and energy savings. SMART [31] and express virtual channel [39] are two techniques that allow packets to bypass the intermediate routers to reduce communication latency dynamically. However, such designs can benefit general-purpose processors with sporadic long-distance communications which fail to well support all the all-reduce collectives. In [40], a reconfigurable link design can dynamically allocate channel bandwidth between adjacent routers, but all-reduce collectives have high bandwidth demands in both link directions. In [41], a reconfigurable NoC changes NoC topology to detour traffic away from the power-gated routers, but the intra-router connectivity is still restricted. SIGMA [42] exploits the flexible interconnect designs at the PE level to tackle diverse dataflows as well as sparse and irregular DNN computations. Such a reconfigurable design focuses on inter-router connections, in which intra-router flexibility remains unexplored.

VII. Conclusion

In this paper, we propose ARIES, an algorithm and architecture co-design in chiplet-based systems that can efficiently support various collective communications in parallel DNN training. Specifically, we propose a flexible NoC design that adapts to various collective communication patterns - ring and tree all-reduce. The proposed NoC design is also capable of adapting to various network topologies, such as hierarchical rings and trees. Moreover, we leverage the NoC flexibility to facilitate existing ring and tree-based all-reduce operations with the aim of improving performance and energy efficiency. Simulation results show that the proposed ARIES can achieve up to 3.92× speedup in execution time and 38.8% reduction in Network-on-Chip (NoC) energy consumption when compared to prior work.

REFERENCES

[1] J. Devlin, M. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *arXiv preprint arXiv:1810.04805*, 2019.

[2] M. Chen, A. Radford, J. Wu, H. Jun, P. Dhariwal, D. Luan, and I. Sutskever. Generative pretraining from pixels. In *In Proceedings of International Conference on Machine Learning (ICML)*, pages 1691–1703. ACM, 2020.

[3] D. Elbrächter, P. Grohs, A. Jentzen, and C. Schwab. Dnn expression rate analysis of high-dimensional pdes: Application to option pricing. In *Constructive Approximation*, pages 3–71, 2018.

[4] Z. Hu, Y. Zhao, and M. Khushi. A survey of forex and stock price prediction using deep learning. In *Applied System Innovation*, 2021.

[5] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, Q. Le, M. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, and A. Ng. Large scale distributed deep networks. In *In Proceedings of Neural Information Processing Systems (NeurIPS)*, pages 1223–1231. ACM, 2012.

[6] W. Fedus, B. Zoph, and N. Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. In *The Journal of Machine Learning Research*, pages 5232–5270, 2022.

[7] Y. Shao, J. Clemons, R. Venkatesan, B. Zimmer, M. Fojtik, N. Jiang, B. Keller, A. Klinefelter, N. Pinckney, P. Raina, S. Tell, Y. Zhang, W. Dally, J. Emer, C. Gray, B. Khailany, and S. Keckler. Simba: Scaling deep-learning inference with multi-chip-module-based architecture. In *Proceedings of IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 14–27. IEEE, 2019.

[8] Y. Li, A. Louri, and A. Karanth. Spacx: Silicon photonics-based scalable chiplet accelerator for dnn inference. In *Proceedings of IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 831–845. IEEE, 2022.

[9] S. Rashidi, S. Sridharan, S. Srinivasan, and T. Krishna. Astra-sim: Enabling sw/hw co-design exploration for distributed dl training platforms. In *Proceedings of IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 81–92. IEEE, 2020.

[10] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778. IEEE, 2015.

[11] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *arXiv preprint arXiv:1409.1556*, 2015.

[12] M. Naumov, D. Mudigere, H. Shi, J. Huang, N. Sundaraman, J. Park, X. Wang, U. Gupta, C. Wu, A. Azzolini, et al. Deep learning recommendation model for personalization and recommendation systems. In *arXiv preprint arXiv:1906.00091*, 2019.

[13] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Proceedings of International Conference on Neural Information Processing Systems (NeurIPS)*, pages 6000–6010. ACM, 2017.

[14] P. Sanders, J. Speck, and J. Träff. Two-tree algorithms for full bandwidth broadcast, reduction and scan. In *Parallel Comput.*, pages 581–594, 2009.

[15] J. Huang, P. Majumder, S. Kim, A. Muzahid, K. Yum, and E. Kim. Communication algorithm-architecture co-design for distributed deep learning. In *Proceedings of ACM/IEEE International Symposium on Computer Architecture (ISCA)*, pages 181–194. IEEE, 2021.

[16] S. Naffziger, K. Lepak, M. Paraschou, and M. Subramony. 2.2 amd chiplet architecture for high-performance server and desktop products. In *Proceedings of IEEE International Solid- State Circuits Conference (ISSCC)*, pages 44–45. IEEE, 2020.

[17] T. Li, J. Hou, J. Yan, R. Liu, H. Yang, and Z. Sun. Chiplet heterogeneous integration technology—status and challenges. In *Electronics*, 2020.

[18] Y. Li, J. Park, M. Alian, Y. Yuan, Z. Qu, P. Pan, R. Wang, A. Schwing, H. Esmaeilzadeh, and N. Kim. A network-centric hardware/algorithm co-design to accelerate distributed training of deep neural networks. In *Proceedings of IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 175–188. IEEE, 2018.

[19] Y. Bai, C. Li, Q. Zhou, J. Yi, P. Gong, F. Yan, R. Chen, and Y. Xu. Gradient compression supercharged high-performance data parallel dnn training. In *Proceedings of ACM Symposium on Operating Systems Principles (SIGOPS)*, pages 359–375. ACM, 2021.

[20] M. Zinkevich, M. Weimer, L. Li, and A. Smola. Parallelized stochastic gradient descent. In *Proceedings of International Conference on Neural Information Processing Systems (NeurIPS)*, pages 2595–2603. ACM, 2010.

[21] C. Ying, S. Kumar, D. Chen, T. Wang, and Y. Cheng. Image classification at supercomputer scale. In *arXiv preprint arXiv:1811.06992*, 2018.

[22] T. Ben-Nun and T. Hoefler. Demystifying parallel and distributed deep learning: An in-depth concurrency analysis. In *ACM Computing Surveys (CSUR)*, pages 1–43, 2019.

[23] P. Patarasuk and X. Yuan. Bandwidth optimal all-reduce algorithms for clusters of workstations. In *Journal of Parallel and Distributed Computing*, pages 117–124, 2009.

[24] U. von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. https://github.com/baidu-research/baidu-allreduce, 2017. [Online; accessed February 2, 2023].

[25] NVIDIA. Nvidia collective communications library (NCCL). https://developer.nvidia.com/nccl, 2016. [Online; accessed February 2, 2023].

[26] G. Wang, S. Venkataraman, A. Phanishayee, N. Devanur, J. Thelin, and I. Stoica. Blink: Fast and generic collectives for distributed ml. In *Proceedings of Machine Learning and Systems (MLSys)*, pages 172–186. mlsys.org, 2020.

[27] B. Klenk, N. Jiang, G. Thorson, and L. Dennison. An in-network architecture for accelerating shared-memory multiprocessor collectives. In *Proceedings of ACM/IEEE Annual International Symposium on Computer Architecture (ISCA)*, pages 996–1009. IEEE, 2020.

[28] H. Zheng, K. Wang, and A. Louri. Adapt-noc: A flexible network-on-chip design for heterogeneous manycore architectures. In *Proceedings of IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 723–735. IEEE, 2021.

[29] H. Zheng, K. Wang, and A. Louri. A versatile and flexible chiplet-based system design for heterogeneous manycore architectures. In *Proceedings of ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020.

[30] H. Zheng and A. Louri. Agile: A learning-enabled power and performance-efficient network-on-chip design. *IEEE Transactions on Emerging Topics in Computing*, 10(1):223–236, 2020.

[31] O. Chen, S. Park, T. Krishna, S. Subramanian, A. Chandrakasan, and L. Peh. Smart: a single-cycle reconfigurable noc for soc applications. In *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 338–343. IEEE, 2013.

[32] P. Sanders, J. Speck, and J. Träff. Two-tree algorithms for full bandwidth broadcast, reduction and scan. In *Parallel Computing*, pages 581–594, 2009.

[33] S. Jeaugey. Massively scale your deep learning training with nccl 2.4. In *NVIDIA Developer Blog*, 2019.

[34] J. Huang, P. Majumder, S. Kim, A. Muzahid, K. Yum, and E. Kim. Communication algorithm-architecture co-design for distributed deep learning. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, pages 181–194. IEEE, 2021.

[35] J. Kim, J. Balfour, and W. Dally. Flattened butterfly topology for on-chip networks. In *Proceedings of IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 172–182. IEEE, 2007.

[36] A. Mirhosseini, M. Sadrosadati, B. Soltani, H. Sarbazi-Azad, and T. Wenisch. Binochs: Bimodal network-on-chip for cpu-gpu heterogeneous systems. In *Proceedings of IEEE/ACM International Symposium on Networks-on-Chip (NOCS)*, pages 1–8. IEEE, 2017.

[37] Y. Yao and Z. Lu. iNPG: Accelerating critical section access with in-network packet generation for noc based many-cores. In *Proceedings of IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 15–26. IEEE, 2018.

[38] J. Yang, H. Zheng, and A. Louri. Venus: A versatile deep neural network accelerator architecture design for multiple applications. In *proceedings of ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023.

[39] A. Kumar, L. Peh, P. Kundu, and N. Jha. Express virtual channels: towards the ideal interconnection fabric. In *Proceedings of ACM/IEEE International Symposium on Computer Architecture (ISCA)*, pages 150–161. IEEE, 2007.

[40] M. Faruque, T. Ebi, and J. Henkel. Configurable links for runtime adaptive on-chip communication. In *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 256–261. IEEE, 2009.

[41] R. Parikh, R. Das, and V. Bertacco. Power-aware nocs through routing and topology reconfiguration. In *Proceedings of ACM/IEEE Design Automation Conference (DCA)*, pages 1–6. IEEE, 2014.

[42] E. Qin, A. Samajdar, H. Kwon, V. Nadella, S. Srinivasan, D. Das, B. Kaul, and T. Krishna. Sigma: A sparse and irregular gemm accelerator with flexible interconnects for dnn training. In *Proceedings of IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 58–70. IEEE, 2020.