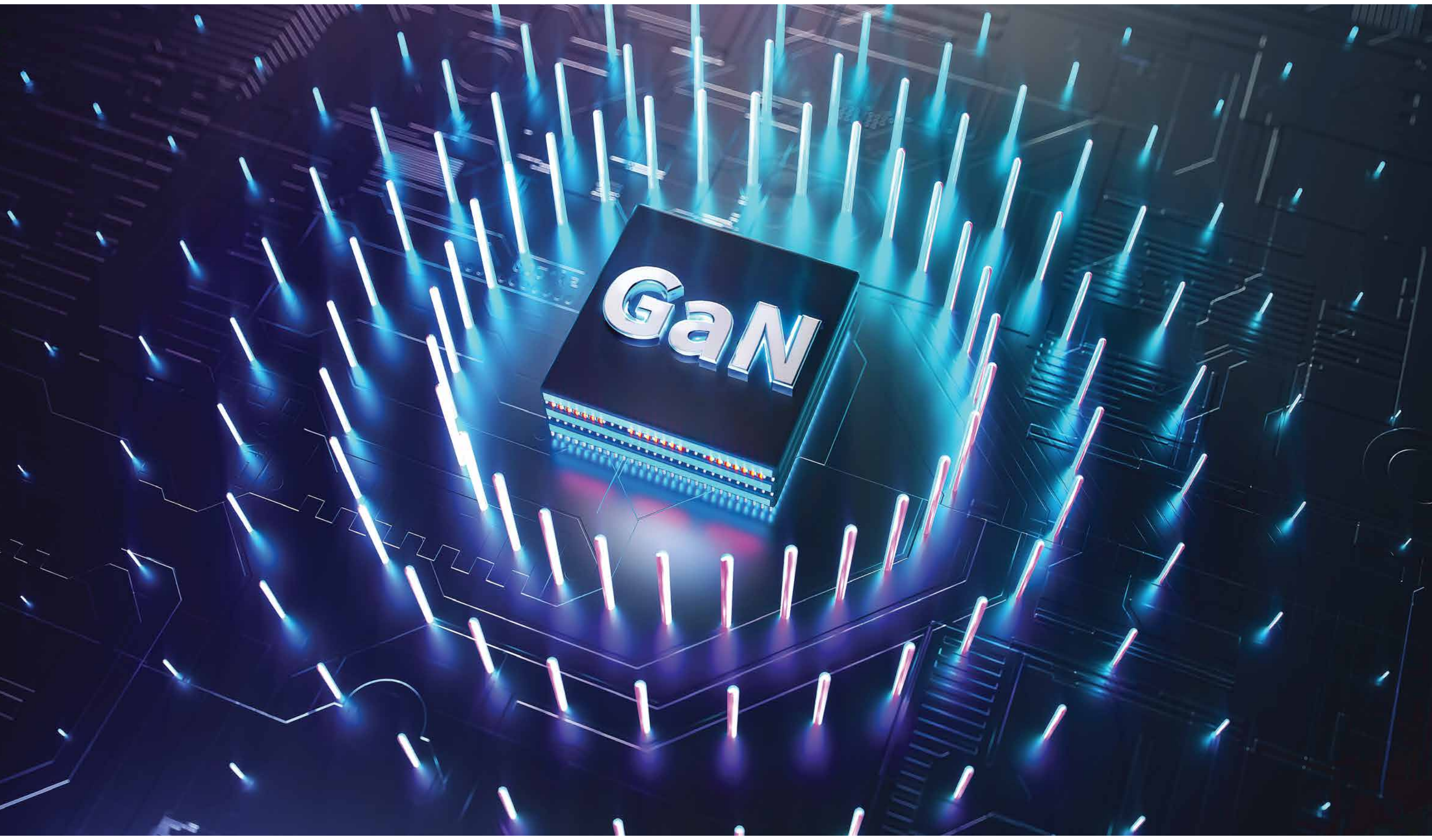

Chiplet-GAN: Chiplet-Based Accelerator Design for Scalable Generative Adversarial Network Inference

Yuechen Chen, *Member, IEEE*, Ahmed Louri, *Fellow, IEEE*, Fabrizio Lombardi, *Fellow, IEEE*, and Shanshan Liu, *Senior Member, IEEE*

Digital Object Identifier 10.1109/MCAS.2024.3359571
Date of current version: 15 August 2024

Abstract

Generative adversarial networks (GANs) have emerged as a powerful solution for generating synthetic data when the availability of large, labeled training datasets is limited or costly in large-scale machine learning systems. Recent advancements in GAN models have extended their applications across diverse domains, including medicine, robotics, and content synthesis. These advanced GAN models have gained recognition for their excellent accuracy by scaling the model. However, existing accelerators face scalability challenges when dealing with large-scale GAN models. As the size of GAN models increases, the demand for computation and communication resources during inference continues to grow. To address this scalability issue, this article proposes Chiplet-GAN, a chiplet-based accelerator design for GAN inference. Chiplet-GAN enables scalability by adding more chiplets to the system, thereby supporting the scaling of computation capabilities. To handle the increasing communication demand as the system and model scale, a novel interconnection network with adaptive topology and passive/active network links is developed to provide adequate communication support for Chiplet-GAN. Coupled with workload partition and allocation algorithms, Chiplet-GAN reduces execution time and energy consumption for GAN inference workloads as both model and chiplet-system scales. Evaluation results using various GAN models show the effectiveness of Chiplet-GAN. On average, compared to GANAX, SpAtten, and Simba, the Chiplet-GAN reduces execution time and energy consumption by 34% and 21%, respectively. Furthermore, as the system scales for large-scale GAN model inference, Chiplet-GAN achieves reductions in execution time of up to 63% compared to the Simba, a chiplet-based accelerator.

Index Terms—GAN inference, chiplet, scalability, interconnection network, machine learning.

I. Introduction

Generative Adversarial Networks (GANs) have excellent performance in content generation, such as image synthesis and natural language processing (NLP) tasks [1], [2], [3], [4]. With the most recent advancement in training algorithms and models, the latest large-scale GAN models have achieved very good performance [5], [6], [7], [8], [9], [10], [11]. For example, TransGAN [6] has shown excellent accuracy on image synthesis tasks by incorporating transformers into the model. To deploy these GAN models, inference is a quintessential step for the model to generate content with reduced delay [12]. The increasing complexity and size of the large-scale GAN model for capturing more complex patterns and generating higher-quality outputs present challenges in architectural design when it comes to handling the demanding computational and communication workloads associated with model inference.

Existing research has proposed several architectures to accelerate GAN inference [12], [13], [14], [15], [16]. Different from convolutional neural networks, a GAN contains transposed convolution operations (TC) and transformers [17], which demand a significant amount of computation and communication resources. Thus, specialized accelerators are developed to handle these workloads. In GANAX [12], a SIMD-MIMD mixed architecture is developed to handle TC. In SpAtten [13], a specialized unit is designed to accelerate the computation of attention mechanisms in transformers. Existing architectures primarily excel in improving efficiency for specific functionalities, such as TC or the attention mechanism. However, as the parameters and the number of layers increase to expand the widespread usage and accuracy of GAN models, the task of scaling these architectures to meet the growing computational and communication demands becomes an increasingly pressing concern. All existing accelerators [12], [13], [14], [15] are implemented with a global controller and global buffer, which are not scalable to support more computation units for the inference of large-scale GAN models.

Recent research has provided compelling evidence of the advantages of chiplet systems in terms of computation performance and scalability [18], [19], [20], [21]. A chiplet system provides a modular approach to semiconductor design and manufacturing, in which multiple smaller chip components (i.e., chiplets) are integrated into a single system. The chiplets are interconnected via network-on-package (NoP) with either active or passive links on a common interposer [22], [23], [24]. Within each chiplet, a multi-core system is implemented by using network-on-chip (NoC) to connect cores, caches, and memory interfaces. Packets are used to carry data in NoCs, and routers are implemented to direct packets to their destination. With such a design, the system offers scalability by allowing users to scale their systems by adding chiplets as needed. Existing research and commercial products have proven that these systems can integrate a large number of computation units at a lower cost and higher flexibility compared to conventional single-chip designs [20], [22], [24], [25]. The flexibility and scalability are particularly useful in GAN model inference, in which varying levels of performance and functionality are required.

To the best of the authors' knowledge, there has been no prior research conducted on designing a scalable

Yuechen Chen is with the Department of Computer Science and Information Technologies, Frostburg State University, Frostburg, MD 21532 USA (e-mail: ychen@frostburg.edu).
Ahmed Louri is with the Department of Electrical and Computer Engineering, The George Washington University, Washington, DC 20052 USA (e-mail: louri@gwu.edu).
Fabrizio Lombardi is with the Department of Electrical and Computer Engineering, Northeastern University, Boston, MA 02115 USA (e-mail: lombardi@ece.neu.edu).
Shanshan Liu is with the School of Information and Communication Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China (e-mail: ssliu@uestc.edu.cn).

GAN inference accelerator by employing a chiplet design approach. Existing research has proposed the use of a chiplet system for deep convolutional neural networks (DCNN), which mainly consists of convolutional layers [18], [19]. Different from DCNN models, TC and transformers require more computation resources with vastly different communication patterns. Compared to existing single-chip accelerator designs [12], [13], [14], [15] for GAN inference that can efficiently process only one type of layer (e.g., matrix multiplication or attention mechanism), a chiplet system excels at efficiently processing multiple layers in parallel. Furthermore, the scalability inherent to a chiplet design approach ensures efficient inference not only for existing large-scale GAN models but also for future models with increasingly demanding computation and communication requirements.

This article presents Chiplet-GAN, a chiplet-based accelerator designed specifically for scalable and efficient GAN inference. Through a detailed analysis of the computation and communication requirements involved in GAN inference, it is observed that operations such as matrix multiplication, reduction operations, up-sampling, and matrix reshape operations consume a significant portion of the inference time and incur intensive communication and computation. Recognizing the distinct communication patterns exhibited by these operations, an adaptive network topology is developed to effectively address the communication demands of the workload. This adaptive interconnection network is implemented in each chiplet, enabling dynamic switching between a concentrated mesh (C-Mesh) and a mesh topology based on the functionality of the layer. For efficient communication between chiplets as Chiplet-GAN scales, NoP is implemented with both passive and active network links on the interposer to efficiently handle both short-distance and long-distance communications. Together with the NoP design and adaptive interconnection network, the efficiency of traffic forwarding is improved during GAN inference, specifically for the aforementioned operations. To complement the adaptive network topology and passive/active network links, the workload partitioning and allocation algorithms are developed. These algorithms partition the workload and strategically distribute the workloads to ensure low communication latency, which further reduces the overall execution time. The key contributions of this article are then summarized as follows.

- This article proposes Chiplet-GAN, a scalable and efficient GAN inference accelerator, by employing a chiplet design approach.
- Chiplet-GAN includes an adaptive interconnection network in chiplet to ensure low communication latency during inference as the GAN model scales.
- Network-on-packet (NoP) is designed with both active and passive network links on the interposer with a workload partitioning and allocation algorithm to ensure low communication latency between chiplets as the system scales.
- Simulation results show that compared to GANAX, SpAtten, and Simba, Chiplet-GAN reduces execution time and energy consumption by 34% and 21%, respectively. Furthermore, as the system scales for large-scale GAN inference, Chiplet-GAN achieves reductions in execution time of up to 63% compared to the existing chiplet-based accelerator (Simba).

II. Motivation and Challenges

A. GAN Inference

The recent large-scale GAN model for inference poses challenges mainly due to the deep model architecture with a large number of parameters and activations. The main challenges of the scaling of the layers that are widely used in GAN models are listed as follows.

- **Fully Connected/Linear/Convolution Layer:** These layers involve matrix multiplications between the input activations and the learned weight matrices. These matrix multiplication operations are computationally expensive, especially when large input activations and weights are used in the model.
- **Attention Mechanism:** The attention mechanism, commonly used in transformer-based GANs, requires pairwise computations between all positions in the input activations. This involves computing attention weights for each position, which entails a significant amount of computation due to many matrix multiplications. Also, the transformer involves matrix reshape operations, such as matrix transpose, and incurring extensive communication traffic between computation units.
- **Up-Sampling Operations:** GAN models utilize up-sampling operations, such as zero insertion or nearest-neighbor up-sampling. Up-sampling operations involve interpolating or expanding input activations to increase the spatial resolution, so requiring extensive data communication between computation units.
- **Activation Functions:** The application of nonlinear activation functions, such as *ReLU* (Rectified Linear Unit), and linear activation functions, such as *SoftMax* and normalization in the model, also contributes to the complexity of computation and communication. These activation functions

are applied elementwise to the activations and require additional computation. The normalization and *SoftMax* functions need to survey the inputs before an element-wise operation, that requires a reduction operation prior to an elementwise operation.

As the GAN model keeps scaling to capture more complex patterns and generate higher-quality outputs, efficient inference in such a model requires more communication and computation capabilities for storing, accessing, and processing input activations and parameters. Thus, a scalable system is needed to handle the increasing demand in both computation and communication for GAN inference.

B. Chiplet Design Approach

Chiplet systems offer significant advantages in terms of performance and scalability for architecture design and semiconductor manufacturing [18], [19], [20], [21], [25]. By integrating specialized chip components, known as chiplets, into a unified system, designers can better utilize each chiplet for specific tasks, resulting in enhanced performance. The modular nature of chiplet systems further enables scalability, allowing for nearly effortless addition or removal of chiplets as required. This flexibility allows system upgrades for large-scale GAN model inference workloads.

To support easy system upgrade and scaling, a network-on-packet (NoP) is implemented to connect chiplets. As shown in Fig. 1, the connection between chiplets is accomplished with a passive or active network link via micro-bump (μbump). Recent work [24] has studied both designs by analyzing the positive and negative of the two designs. Active network links enable better scalability compared to passive network links with larger throughput for long-distance communication. However, implementing active network links needs routers being implemented in the interposer, which requires the packet to traverse routers in both chiplet and interposer. On the other hand, passive network links achieve lower network latency by only relying on the router in the chiplet, even though the link has lower bandwidth.

Within each chiplets, processing elements (PEs), caches, and memory interfaces are connected using NoC. All the data is transmitted using packets in both NoC and NoP. Overall, chiplet systems present a promising approach to improve performance, efficiency, and scalability in GAN inference accelerator designs.

As chiplet systems rely heavily on the interconnection network to connect and communicate between different computation units and memories, designing an efficient interconnection network poses a major challenge for efficient and scalable GAN inference. Energy and latency represent two vital aspects when designing interconnect solutions for efficient GAN inference. With the addition of more chiplets to accommodate the expanding GAN model, the NoP inevitably becomes larger; consequently, the traversal of packets through the expanded network incurs higher time and energy costs. According to the existing research [18], [19], [20], [24], [25], network latency and energy consumption significantly impact inference efficiency with longer execution time and high energy consumption. Thus, reducing latency and energy consumption for interconnection networks is crucial for GAN inference on a chiplet system, which often requires real-time responsiveness during deployment. Specifically, the main challenges for designing a chiplet system for GAN inference are listed below.

- **Diverse Communication Pattern:** Inference involving various layers in GAN models results in markedly diverse communication patterns. Given the significant effects of latency and power consumption when a packet is redirected by the routers, there's a pressing need for innovative NoC designs, which should efficiently address the distinct communication patterns observed during GAN inference within a chiplet.
- **Scalable and Efficient NoP:** The communication between chiplets is a major bottleneck for chiplet-based design as the system scales. One major challenge in NoP design is the usage of passive and active network links. Existing research [24] indicates that passive links can handle high data throughput over short distances. However, their scalability is constrained by limited communication range. Conversely, active links facilitate long-distance communication, but their bandwidth is restricted due to the incorporation of routers on the interposer. Considering the benefits and limitations of both active and passive

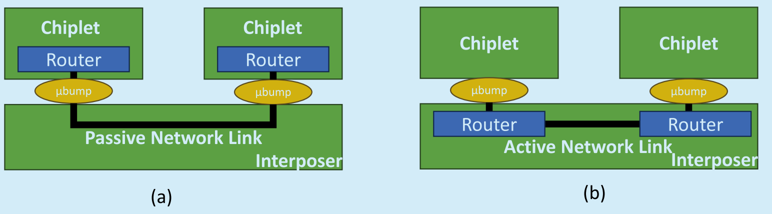

Figure 1. NoP design with a passive or active network link. (a) NoP with passive network link. (b) NoP with active netwrok link.

links, a novel NoP design, grounded in a strategic analysis of GAN inference workloads, is crucial for the efficiency of the chiplet-based accelerator. Additionally, the network-on-package (NoP) should be synergistically designed alongside workload partitioning and allocation algorithms, aiming to harness the strengths of both interposer types.

III. Chiplet-GAN Design

A. Overview

The goal of the proposed accelerator design is to attain both improvements in efficiency and scalability for GAN inference workloads by employing a chiplet design approach. This is mainly accomplished through the following steps.

- A thorough analysis of the computation and communication requirements specific to the GAN model is conducted. This analysis involves examining the architecture of various GAN models used for image synthesis workloads and meticulously assessing the computation and communication demands of each layer as well as the overall model.
- Chiplet-GAN is proposed, featuring an adaptive network topology design for the interconnection network to facilitate the scaling of individual layers.
- Workload partition and allocation algorithms are introduced to effectively utilize the benefits of the adaptive topology design when dealing with large-scale GAN models.

B. Layer Analysis

Table 1 shows the list of large-scale GAN models used in this article. These models contain tens of millions of parameters and multiple layers to generate high-resolution images depending on the configuration. Additionally, newer variations of these models contain more parameters to achieve better image resolution. The basic structure of these models includes both transposed convolution (TC) and transformers, which are the two design trends for the GAN model. The TC-based model includes convolution, up-sampling functions, normalization functions, and activation functions. The transformer-based model contains attention mechanisms, linear layers, up-sampling functions, normalization functions, and activation functions. The functionality of these layers consists of several basic operations, including matrix multiplication, reduction operation, and up-sampling/matrix reshape operations. Communication and computation are needed when computation units execute these basic operations. The analysis of the computation and communication requirements for these basic operations is presented next.

Table 1.
GAN models.

GAN Models	Basic Structure	Number of Parameters	Task
BigGAN-deep [9]	TC	50 million	Image synthesis
ViTGAN-Base [11]	Transformer	38 million	Image synthesis
TransGAN-Base [6]	Transformer	85 million	Image synthesis
StyleGAN2 [5]	TC	24 million	Text-to-Image
CycleGAN [7]	TC	35 million	Image-to-Image Translation

1. Matrix Multiplication

Fully connected, linear, and convolution layers are widely used in both transformer and TC-based GANs. In a transformer, the linear layers are widely used before the attention mechanism. Also, the transformer utilizes matrix multiplication to calculate attention values. In TC-based GANs, the linear layer is used at the end of each processing block to capture the complex relationship between input and output data. Eq. (1) shows the basic operation for a fully connected layer and linear layer, i.e., matrix multiplication. The convolution layer can also be accomplished using matrix multiplication by reshaping the input activation and weight matrices.

$$O_{mp} = A_{mn} * B_{np} \tag{1}$$

Eq. (1) shows two input matrices (i.e., A, B) and one output matrices (i.e., O). m, n denotes the number of rows and columns of matrix A, and n, p denotes the number of rows and columns of matrix B. To perform matrix multiplication, the computation unit needs to load matrices A and B, which requires data communication of $m * n + n * p$ elements. Then, the matrix multiplication requires n $* m * p$ multiplications followed by $(n-1) * m * p$ addition operations. After the computation, the computation unit generates and sends out the matrix O with $m * n$ elements.

2. Reduction Operation

Both *SoftMax* and the normalization layer scale the input matrix into a specific format based on the statistics of the input activations. Since these layers improve the accuracy of the model, they are widely used in both transformer and TC-based GANs. These layers typically contain two steps. The first step calculates the statistics of the input, which involves a reduction operation. Eqs. (2) and (3) show the function for statistic computation for *SoftMax* and normalization operations.

$$s = \sum_{j=1}^{i} e^{x_j} \tag{2}$$

$$\begin{cases} \mu = \frac{1}{i}\sum_{j=1}^{i} x_j \\ \sigma^2 = \frac{1}{i}\sum_{j=1}^{i} (x_j - \mu)^2 \end{cases} \tag{3}$$

In Eqs. (2) and (3), the input matrix x contains a total of i elements. Both functions need to access all elements in the input matrix to get the statistics. Assume the input matrix has a size of $m * n$. *SoftMax* needs data communication of $m * n$ elements to gather all inputs, so $m * n$ exponent computations and $m * n - 1$ additions to get the variable s. For the normalization layer, $m * n$ elements are gathered to calculate the mean value of the inputs through $m * n - 1$ addition and 1 division operations. Then, the variance is calculated through $m * n$ subtraction and square operations, followed by $m * n - 1$ additions and one division. Both of these layers require a reduction operation, which requires data communication, to gather the input activation and calculate the statistics. After the calculation, the element-wise operations are applied to the input matrix.

3. Up-Sampling/Matrix Reshape Operation

Up-sampling operations are widely used in GANs to increase the size of the output matrix, such as zero insertion and nearest neighbor. These layers require a matrix with $m * n$ elements as the input and upscale it a time, which requires $m * n + a * m * n$ communication. The matrix reshape operations, such as transpose, take a matrix with $m * n$ elements as the input and output a matrix with $n * m$ elements. Both up-sampling and matrix reshape operations typically do not require any computation on the input activation, but they require communications to send data to the correct destination. Table 2 summarizes the layer analysis with the computation and communication ratio ($R_{c\&c}$) as defined in

$$R_{c\&c} = \frac{\text{Number of Computation}}{\text{Size of Input and Output Matrice}} \tag{4}$$

In Eq. (4), the amount of communication is estimated by adding the size of both input and output matrices. The layer requires more computation when the $R_{c\&c}$ value is higher. However, a layer requires more communication with a low $R_{c\&c}$ value. By comparing the $R_{c\&c}$ value for different layers, it can be seen that except for the fully connected/linear layer, in which computation is significantly higher than communication, the remaining layers require significantly more communication compared to the fully connected/linear layer.

C. Static GAN Model Analysis

Fig. 2 shows the breakdown of the communication requirements by statically analyzing the GAN models layer by layer in Table 1. The analysis assumes the inference of the GAN model is executed on sixteen chiplets, which contain multiplication and addition arrays for matrix multiplication and computation units for activation functions and up-sampling operations. The workload is evenly divided and distributed by the size of the activation for each layer. Based on the analysis, the reduction operations and up-sampling/reshape operations occupy more than 72% of the total communication. 36% of the communication is due to the reduction operation to calculate the statistics for the *SoftMax*/normalization layer. Thus, the challenges for the inference of large-scale GAN models on a scalable chiplet system are listed below.

1) **Scaling Reduction Operation**. The wide usage of the normalization layer results in significant communication. Most of these communications are due to the reduction operation for statistics calculation and element-wise operation. Moreover, as the size of activation increases for high-resolution image generation, supporting a large-scale reduction operation on chiplet systems becomes a challenge.
2) **Scaling Matrix Multiplication**. As the GAN models scale, the matrix multiplication workloads increase dramatically. To handle the increasing demand in computation for large matrix multiplication, a scalable chiplet system is needed.
3) **Distributed Up-Sampling/Matrix Reshape**. As the GAN models generate high-resolution data, these operations need to process a large activation matrix during inference, which involves significant communication traffic. Since existing accelerators use a scratch pad memory, both operations require a centralized global controller to reorganize

Table 2.
Summary of layer analysis.

Layer Name	$R_{c\&c}$
Fully Connected/Linear	$(n^2\text{-}n)mp/mn+np+mp$
Layer Normalization	$3mn/2mn = 1.5$
SoftMax	$(3mn\text{-}1)/2mn \approx 1.5$
Upscaling/Matrix Reshape	0

the data mapping in the memory, which is not scalable for a chiplet system. For a chiplet system, these operations must be distributed rather than centralized, which incurs a large amount of communication between chiplets for transmitting activations. This communication incurs a high latency, especially when the packet must travel a long distance.

D. Architecture Design

To solve the three challenges for scalable and efficient GAN inference on a chiplet system, the architecture of Chiplet-GAN includes the following features.

- **Adaptive NoC Topology**: To satisfy the communication requirement for the reduction operation, designing a concentrated network allows an efficient gathering of data for the operation. However, such a network has low efficiency for scaling matrix multiplication operations, as matrix multiplication needs frequent data exchange between neighbors. To support both reduction and matrix multiplication operations, Chiplet-GAN utilizes an adaptive network topology design, which dynamically switches between mesh and concentrated mesh (C-Mesh) topologies for efficient on-chip communication. This design ensures the efficiency of the execution of both reduction and matrix multiplication operations.
- **NoP With Both Active and Passive Network Links:** To efficiently handle both long-distance and short-distance communication between chiplets as the system scales, both active and passive network links are implemented in the NoP. Specifically, active network links are implemented for efficient long-distance communication between chiplets and passive network links for low latency communication between two chiplets.
- **Workload Partitioning and Allocation:** To reduce the communication latency for GAN inference, specialized workload partition and allocation algorithms are developed based on static model analysis. The proposed partition and allocation algorithm utilizes the characteristics of both the proposed NoP design and adaptive network topology in NoC for efficient and scalable GAN inference on Chiplet-GAN.

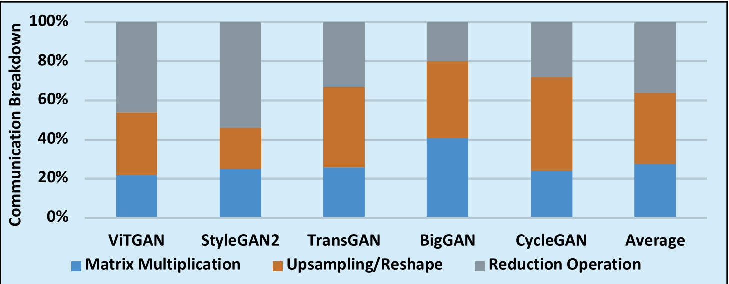

Figure 2. Communication breakdown through static GAN model analysis.

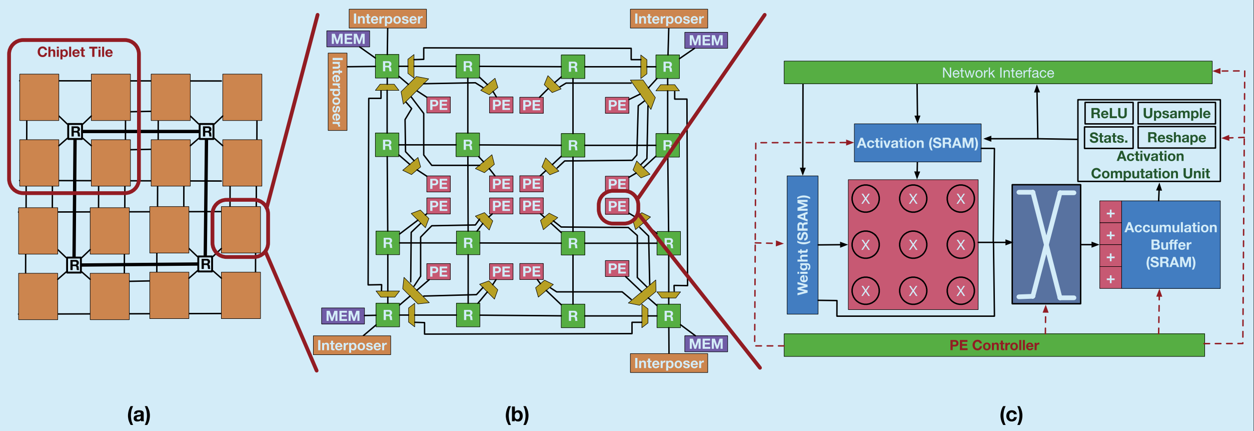

Figure 3. Architecture of Chiplet-GAN. The system package is scalable to include more chiplets tiles. Each chiplet tile consists of four chiplets and is connected to a router (R) on the interposer. The active link is implemented to connect routers for chiplet groups. The passive link is implemented to connect two chiplets. Within each chiplet, an adaptive network is implemented with routers (R) to connect the processing element (PE), the memory interface (MEM), and the active/passive interposer. The passive interposer connects the corner routers of the two adjacent chiplets. Each PE contains a PE controller, SRAMs for storing input and output matrices, a crossbar to forward the product of the multiplier array, and an activation computation unit for processing ReLU, activation statistics (Stats.) calculation, and up-sampling/matrix reshape operations. (a) System package. (b) Chiplet. (c) Processing element (PE).

Fig. 3 shows the architecture of Chiplet-GAN. The system package (Fig. 3(a)) contains multiple chiplets, which is scalable for the inference of large GAN models. As shown in Fig. 3(a), each chiplet group contains four chiplets, which are connected to a router via µbumps. This design handles long-distance communication between chiplets by connecting the corner router on each chiplet and the NoP router in a chiplet group using active network links. These active network links connect chiplets tiles, which not only enables the scaling of the chiplet system with additional tiles but also allows high bandwidth data exchange between tiles. For short-distance communication between adjacent chiplets, passive network links are implemented to enable quick data exchange between the corner routers of the two chiplets.

Fig. 3(b) shows the detailed design of the chiplet. The chiplet includes an adaptive network to efficiently process both reduction operation and matrix multiplication operation by switching between C-mesh and mesh topology. This is achieved by controlling the multiplexers and demultiplexers. By switching to the C-mesh topology, the latency of communication between chiplets is reduced by skipping routers and network links.

Fig. 3(c) illustrates the detailed design of each processing element (PE). Each PE contains an array of multipliers and an accumulative buffer for multiplication and addition operations. The activation computation unit is designed to execute several functions, including ReLU, activation statistics computation, and matrix up-sampling/reshape. Each PE contains a PE controller, which controls the computation process and activation

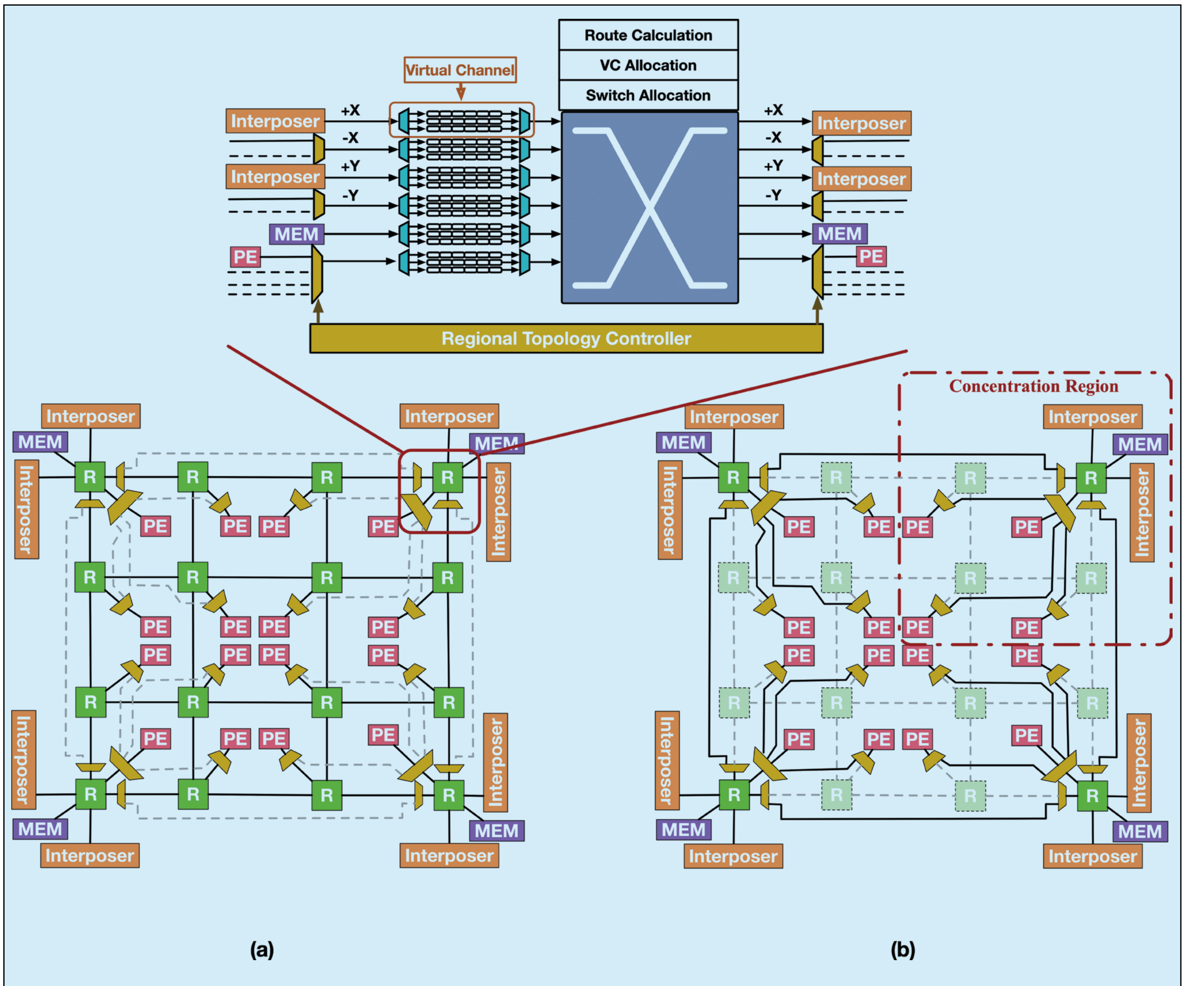

Figure 4. Adaptive network design in Chiplet. All the routers (R) in NoC have the same design with different multiplexer and demultiplexer connections. The dotted line indicates skipped links or routers for communication latency reduction. (a) Mesh topology. (b) C-Mesh topology.

computation unit. The proposed PE utilizes an output-stationary data flow to reduce the frequent exchange of partial sum during the matrix multiplication. With this data flow, the GAN inference workload is partitioned by the size of the output activation of a layer. Section III.E discusses the details of workload partitioning and allocation.

Fig. 4 shows the detailed design of the routers and the two configurations of the topology of NoC in each chiplet. Each chiplet consists of 16 PEs, which can be configured into a 4 * 4 mesh topology (Fig. 4(a)) or into a 2 * 2 C-Mesh topology (Fig. 4(b)). The PEs on each chiplet are partitioned into 4 concentration regions with 4 PEs for each region. The corner router of each chiplet is connected to one memory interface and the interposer to communicate with the neighbor chiplet. The router in Fig. 4 transmits a packet with a five-stage pipeline, which includes route calculation, virtual channel (VC) allocation, switch allocation, switch traversal, and link traversal. The multiplexers and demultiplexers are added between the router and network interface to allow each region to switch to a concentrated topology (i.e., C-Mesh) for efficient data gathering and scattering for PEs in the region. Each region is independently controlled by the regional topology controller, and the topology is changed dynamically during the GAN inference.

To prevent the loss of in-flight packets when switching between the two topologies, the following process is developed in the regional topology controller. After the chiplet completes the inference of one layer, the topology controller prepares to switch topology by monitoring the flits in VC. When the controller observes the last flit in VC is the tail flit, the multiplexers and demultiplexers for that VC are switched immediately. Otherwise, switching of topology is delayed until the tail flit enters the VC. With this switching process, there will be no information in the network link when the controller switches network topology, as the multiplexers and demultiplexers are implemented before the VC. Also, this process ensures the packet is intact when it is traversed through the router.

Compared to the NoP and NoC designs in Simba [18], the Chiplet-GAN facilitates not only efficient communication for reduction and matrix multiplication operations but also low latency communication between chiplets. This is mainly due to the implementation of an active interposer with both active and passive network links and the implementation of the adaptive network to skip excessive links and routers for communication by utilizing the C-Mesh topology. Compared to the passive-only interposer and network link for the NoP in Simba, the proposed NoP design utilizes chiplet tiles and active links for long-distance communication as systems scale and passive links for low latency communication between the adjacent chiplets. Moreover, within each chiplet, the C-Mesh topology also enables quick DRAM access through the NoC as well as easy adapting to mesh topology by only adding several multiplexers and demultiplexers to the conventional NoC routers. Section III.F discusses the configuration process of the adaptive network during the GAN inference.

E. Workload Partitioning and Allocation

Long-distance communication between chiplet impacts the execution time of GAN inference. Although the long-distance communication cost between chiplets can be reduced by utilizing active network links and routers in NoP, the GAN inference workload must be carefully partitioned to fully utilize this feature. Moreover, for layers that require a large amount of data movement (e.g., Matrix Up-Sampling/Reshape) within a short distance, the communication cost between chiplet can be reduced by utilizing the C-Mesh topology in each chiplet and the passive network link between chiplets. Thus, the proposed workload partitioning and workload allocation algorithms are designed to utilize these network features to reduce the communication cost for GAN inference.

1) Workload Partitioning

Since each PE contains an activation computation unit, the workload is first partitioned by the layer functions. Specifically, the GAN model is partitioned by groups, which contain multiple layers with matrix multiplication operations and one layer for activation computation units. Then, the workload for each group is divided and allocated to a set of chiplet tiles depending on the size of the largest activation and the size of the accumulative buffer in each chiplet tile due to the implementation of output-stationary dataflow in each PE.

$$\mathit{Chiplet\ tile\ \#} = \left\lceil \frac{\text{max activation size in the group}}{\text{accumulative buffer size / chiplet tile}} \right\rceil \tag{5}$$

Eq. (5) illustrates the calculation of the number of chiplet tiles needed for the inference of each layer group, which is the ceiling of the maximum activation size divided by the size of the accumulative buffer in each chiplet tile. With the proposed workload partitioning method, the inference of each group is allocated to a set number of chiplet tiles. As the last layer of each group is an activation layer, which involves reduction operation or matrix up sampling and reshape, the NoP router at the center of each chiplet tile is utilized for low latency communication, specifically, for a single chiplet tile, the router in

NoP along with the C-Mesh topology in the four chiplet forms a concentrated network for efficient exchange of the activation statistics during reduction operation. For the group that requires more than one chiplet tile, the active network link in NoP provides low-latency communication between the adjacent chiplet tiles. Moreover, after the inference of the activation layer, the NoP is utilized to transmit the activations to the next chiplet tiles for the inference of the next layer. To reduce the communication distance between the layer groups, an effective workload allocation algorithm is essential for the efficiency of Chiplet-GAN.

Algorithm 1
Workload Allocation

```
def calculate_communication_cost(placed_group):
    $H_i$ = # of NoP routers traversed for input activations of a group
    $S_i$ = the size of input activations
    $H_{tp}$ = # of NoP routers traversed for temporary data
    $S_{tp}$ = the size of temporary data traversed through the active NoP link
    $N_{pl}$ = # of traversed passive NoP link for temporary data
    $H_{tc}$ = # of NoC routers traversed for temporary data
    $S_{tc}$ = the size of temporary data traversed through NoC
    $C_i = T_{rp\_late}*H_i + (S_i/S_f)*T_{fp}$
    $C_{tp} = T_{rp\_late}*H_{tp} + (S_{tp}/S_f)*T_{fp}$
    $C_{tc} = T_{rc\_late}*H_{tc} + (S_{tc}/S_f)*(T_{fc} + L_{pl}*N_{pl})$
    $C = C_i + C_{tp} + C_{tc}$
return C
def find_best_chiplet(group, available_chiplets):
    min_cost = infinity
    best_chiplet = None
    for chiplet in available_chiplets:
        Place group to chiplet
        for placed_group in available_allocation:
            cost = calculate_communication_
              cost(placed_group)
            if cost < min_cost:
                min_cost = cost
                best_chiplet = chiplet
return best_chiplet
def map_groups_to_chiplets(groups, chiplet_tiles):
    mapped_groups = {}
    available_chiplets = chiplet_tiles
    for group in groups:
    best_chiplet = find_best_chiplet(group, available_chiplets)
    mapped_groups[group] = best_chiplet
    available_chiplets.remove(best_chiplet)
return mapped_groups
GAN Model M = [group_1,group_2, … , group_x]
Chiplet System C = [chiplet_tile_1, chiplet_tile_2, …,
    chiplet_tile_n]
Allocated_Group = map_group_to_chiplets(M,C)
```

2) Workload Allocation

With the number of chiplet needed for each group, Algorithm 1 is developed to allocate the workloads on the chiplet system. The greedy algorithm is used in the allocation process to reduce the communication latency between chiplets. According to the workload partition, each layer group is assigned more than one chiplet tile. Thus, the allocation algorithm first calculates the communication cost for all the possible allocations for each layer group, then allocation with the minimal communication cost is selected.

As shown in Algorithm 1, suppose the GAN model *M* is partitioned into *x* groups (i.e., *M = [group_1, group_2, …, group_x]*) and the system *C* has *n* chiplets tiles (i.e., *C = [chiplet_tile_1, chiplet_tile_2, …, chiplet_tile_n]*). The *Allocated_Group* stores the results of the workload allocation. In the *map_groups_to_chiplets* function, the function selects and allocates the inference workload to the chiplet tiles group by group. The set for the chiplets is selected by the *find_best_chiplet* function, where the minimal communication cost for all the possible allocations is found by evaluating the communication cost for all the possible workload allocations. The *calculate_communication_cost* function calculates the time needed for transmitting data through the network. According to the static workload analysis, the two types of data that occupy the majority of the traffic are input activations and temporary data, which include activation statistics and partial sums. These data are transmitted through either NoP or NoC during the inference. Thus, the communication cost (C) includes the network latency of input (C_i) and the network latency of temporary data (C_{tp} and C_{tc}). Considering the communication for temporary data can go through NoC and NoP, the C_{tp} and C_{tc} represent the communication latency for NoP and NoC, respectively.

Specifically, after the allocation of the group, the input activation is transmitted from another chiplet tile, which relies on the active links in NoP to transmit activation. The network latency of input activation (C_i) is calculated based on the physical delay of the NoP through the active link (T_{rp_late}), the number of NoP routers traversed (H_i), the size of one flit in a packet (S_f), the size of input activations (S_i), and transmission time for one flit in NoP (T_{fp}). In this function, T_{rp_late} is the physical delay caused by the network's physical aspects, which include switch and link delays. Notability, the latency for NoC communication is constant regardless of the allocation of the workload in the function for the network latency calculation of input activation (C_i), as all the activation data have to traverse the NoC in that chiplet to reach the PEs for inference. Thus, it is assumed that the communication latency in NoC is the same, and only the communication cost for NoP is calculated.

Considering the difference in design for the links in NoC and NoP, the T_{rp_late} and T_{rc_late} represent the physical delay for active links in NoP and NoC, respectively, for the calculation of C_{tp} and C_{tc}. The calculation of C_{tp} includes the network latency for the data traversed through the active link in the NoP, where T_{rp_late} is used. In terms of the calculation of C_{tc}, the network latency includes two parts, namely, the latency of traversing NoC and passive NoP links. When a layer group is allocated on more than one chiplet, the communication in the NoC is short-distance communication; thus, passive NoP links are utilized to transmit temporary data. As the change in workload allocation affects the number of passive NoP links traversed during the transmission of temporary data, the network latency of traversing both NoC and passive NoP links has to be counted. In the calculation of the network latency for NoC, T_{fc} represents the transmission time for one flit in NoC. In the calculation of the network latency for traversing passive NoP links, the L_{pl} represents the transmission time for one flit in passive NoP links. Notability, for the NoC communication cost, the algorithm calculates the communication latency under the C-mesh topology configuration, which is lower compared to mesh topology. The following section illustrates the configuration of the adaptive network in chiplet, which maintains low latency communications during GAN inference.

F. Adaptive Network Configuration

The network topology in each chiplet is dynamically changed depending on the function and the allocation of the layers to the accelerator. The regional topology controller for each concentration region changes the network topology to support the partitioned and placed workload for efficient communication between PEs and between chiplet. The controller selects C-Mesh topology for DRAM access, chiplet-to-chiplet communication, and reduction communication. Thus, the communication latency for chiplet-to-chiplet communication is calculated for the C-Mesh topology in each chiplet. For PE-to-PE communication, the mesh topology is selected. The controller monitors the computation process of four PEs in the concentration region. When the PEs in the region finish one layer, the controller selects the next topology based on the communication requirement for the next layer. The configuration process for matrix multiplication, reduction operation, and up-sampling/matrix reshape operations are listed as follows.

1) **Matrix Multiplication.** The NoC is configured into the C-Mesh topology when loading the SRAM in each PE. Then, the network is configured into mesh for efficient exchange of data during the matrix multiplication. For the matrix multiplication workloads, which are allocated to more than two chiplets, the network is configured into a C-Mesh topology for efficient chiplet-to-chiplet communication or DRAM access after the calculation of the local partial sum.
2) **Reduction Operation.** The NoC is configured into C-Mesh topology for reduction operation. For reduction operation on a single chiplet, the parameter is calculated at the four corner PEs. For reduction operation across multiple chiplets, the corner PEs in each chiplet with the most connection to the neighbor in the previous layer are selected for calculating the parameter. Then, the parameters are sent back to PEs for the next operation through the concentrated network.
3) **Up-Sampling/Matrix Reshape.** For the matrix reshape operation, which needs to either store the activation to DRAM or communication between chiplets, the NoC is configured into the C-Mesh topology for efficient communication.

IV. Evaluation

A. Simulation Setup

In this section, the performance of Chiplet-GAN is evaluated by using the SMAUG [26] simulator. The cycle-accurate SMAUG simulation model is modified to support

Table 3.
Simulation setup.

PE Parameter	Value
Multiplier	32 bits FLOAT
Accumulator	32 bits FLOAT
Number of Multiplier	16
Number of Adder	16
Accumulation Buffer	144*32 bit
Activation/Weight SRAM Size	244*32 bit
Chiplet Parameter	Value
Number of PE	4*4
Link width	512 bit
Packet Size	4 flit * 512 bit
Payload/Packet	3 flit * 512 bit
Accelerator Parameters	Value
Number of Chiplets	4*4
DRAM Bandwidth	1024 GB/s
DRAM Size	32 GB DDR4
Frequency	2 GHz

Chiplet-GAN. Table 3 shows the settings for the SMAUG simulator for the Chiplet-GAN. PyTorch is used with the SMAUG simulator to control the entire GAN inference process. The accelerators are synthesized using Synopsys Design Compiler with TSMC 16nm to obtain power dissipation during GAN inference and area consumption of the system.

Chiplet-GAN is compared against GANAX [12], SpAtten [13], and Simba [18] on execution time, communication latency, energy consumption, area, and scalability. GANAX is a GAN inference accelerator for TC acceleration with a single-chip design. SpAtten is an accelerator designed to accelerate the inference of transformers, which also incorporates a single-chip design. Simba is a chiplet-based accelerator that focuses on the DCNN model inference. All accelerators are implemented with the same number of computation units as well as the same size of SRAM, DRAM, and DRAM bandwidth. The inference of the models listed in Table 1 is executed on these accelerators to generate two images during evaluation.

B. Execution Time

Fig. 5 shows the execution time reduction normalized to Chiplet-GAN. The proposed design reduces execution time by 27%, 29%, and 45% on average, compared to GANAX, SpAtten, and Simba, respectively. Compared to Chiplet-GAN, SpAtten enhanced the efficiency of inference attention mechanism, but it takes a longer time to process up-sampling operations. Thus, Chiplet-GAN achieves a 16% execution time reduction on average for transformer-based GANs (i.e., ViTGAN and TransGAN). For TC-based models (i.e., StyleGAN2, BigGAN, and CycleGAN), which require more frequent up-sampling operations, Chiplet-GAN reduces 38% in execution time on average compared to the SpAtten. Compared to GANAX, which is optimized for up-sampling and matrix reshaping, the Chiplet-GAN achieves a 21% execution time reduction on average.

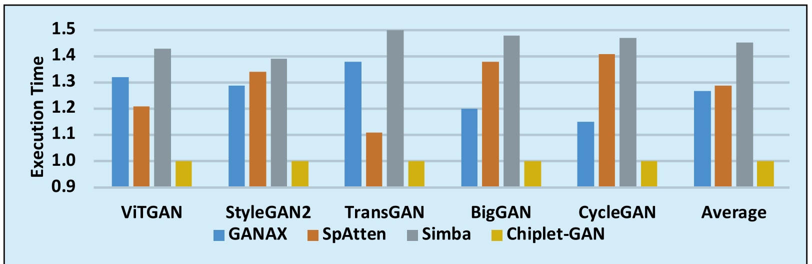

Figure 5. Execution time. Results are normalized to Chiplet-GAN.

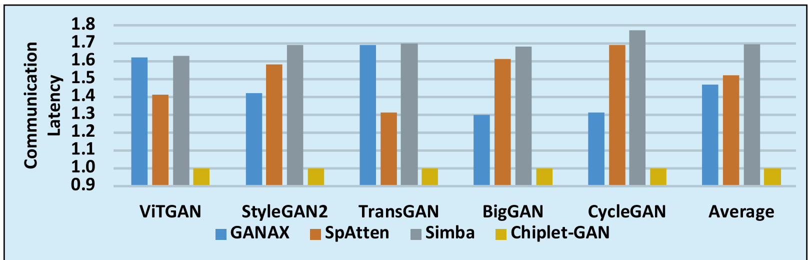

Figure 6. Communication latency. Results are normalized to Chiplet-GAN.

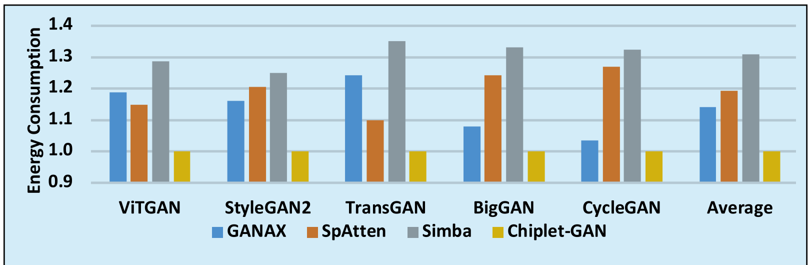

Figure 7. Energy consumption. Results are normalized to Chiplet-GAN.

The main reason for execution time reduction is the implementation of the adaptive interconnection network design, active/passive network link in NoP, and workload allocation strategy. These design features not only reduce the communication latency but also increase the utilization of computation units with fewer idle cycles. The reduction in communication latency is shown in Fig. 6, which is measured by the time elapsed between the access of the data and the beginning of the computation. Chiplet-GAN reduces communication latency by 46%, 52%, and 68% on average, compared to GANAX, SpAtten, and Simba, respectively.

Since both Simba and GANAX mainly focus on accelerating convolutional operations, the up-sampling/reshape and reduction operations are not fully accelerated when executing GAN inference. Specifically, TC acceleration, which contains zero insertion optimization, is incorporated in GANAX design. However, a notable drawback in GANAX is the lack of support for reduction operations. This limitation leads to the need for temporary storage of activations in both the DRAM and global buffer, resulting in a considerable increase in data communication

Chiplet-GAN features an adaptive topology design for the interconnection network and incorporates workload partition and allocation algorithms to efficiently handle large-scale GAN models.

and communication latency for GANAX, especially for the models that require frequent reduction operations (e.g., ViTGAN and TransGAN). As shown in Fig. 6, the GANAX and Simba achieve similar communication latency compared to Chiplet-GAN for ViTGAN and TransGAN.

On the other hand, Simba only focuses on the acceleration of DCNN inference. Thus, the lack of communication latency reduction techniques for both up-sampling/reshape and reduction operations incurs more communication delays and longer execution time. As shown in Fig. 6, Simba incurs 23% and 17% more communication latency compared to GANAX and SpAtten, respectively. Apart from lacking support for the operations during GAN inference, the long-distance communication between chiplets is another issue for Simba. This is mainly caused by the close placement of routers and passive links only NoP design, which further increases the communication latency. As a result, compared to Chiplet-GAN, the Simba incurs significantly longer execution times during GAN inference.

C. Energy Consumption

Fig. 7 shows the evaluation results for the energy consumption of Chiplet-GAN, GANAX, SpAtten, and Simba. All results are normalized to Chiplet-GAN. The energy is the product of execution time and power dissipation. Power dissipation includes two parts: static power and dynamic power.

Chiplet-GAN reduces energy consumption by 20% on average compared to the existing accelerators. Specifically, compared to GANAX, SpAtten, and Simba, Chiplet-GAN reduces energy consumption by 14%, 19%, and 30% on average, respectively. Compared to the reduction in execution time, the energy reduction is less; this is mostly due to the extra power consumed by the adaptive interconnection network in Chiplet-GAN and the active links implemented in the interposer. The GANAX and SpAtten have simplified networks with the predetermined data flow; however, fixed data flow in existing designs results in an unscalable architecture, which significantly impacts execution time as the system scales.

D. Area Evaluation

Table 4 summarizes the area required for the accelerators with the configuration given in Table 3. The results are from the synthesis report of the Synopsys Design Compiler.

Overall, Chiplet-GAN occupies less area compared to SpAtten despite the additional area needed for the connection between chiplets and interposer. This is mainly due to the implementation of a crossbar for on-chip communication, which incurs a large on-chip area in SpAtten. Compared to SpAtten, the proposed

Table 4.
Area.

	GANAX	SpAtten	Simba	Chiplet-GAN
Area (mm2)	5.98	7.48	6.03	6.34

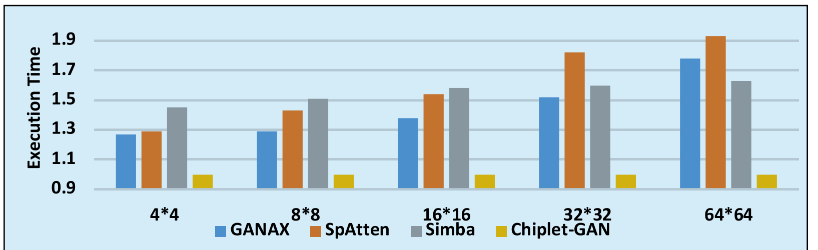

Figure 8. Execution time as the systems scales. Results are normalized to Chiplet-GAN.

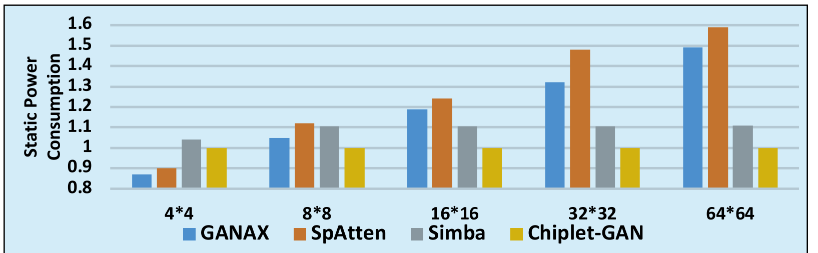

Figure 9. Static power consumption as the systems scales. Results are normalized to Chiplet-GAN.

accelerator reduces the area by 15%. Compared to Simba and GANAX, Chiplet-GAN requires 5% and 6% more area, respectively. This is mainly due to the additional hardware needed to support adaptive interconnection networks and active/passive network links. However, when considering the significant reduction in execution time and energy consumption, the increase in area is rather marginal for implementing the proposed accelerator.

E. Scalability

To demonstrate the scalability of the system, the GAN models are scaled by increasing the size of both activation and weights as Chiplet-GAN scales to 8 * 8, 16 * 16, 32 * 32, and 64 * 64 chiplets. For example, a layer in a model with a weights size of 4 * 4 * 3 and activations size of 128 * 128 * 3 is executed on the system with 4 * 4 chiplets. The weights and activations are scaled to sizes of 8 * 8 * 3 and 256 * 256 * 3, respectively, as the system scales from 4 * 4 to 8 * 8. The DRAM size and DRAM bandwidth are scaled accordingly by doubling the size and bandwidth to handle the significant increase in memory port and memory traffic. The SpAtten, GANAX, and Simba are also scaled accordingly with the same amount of computation units and on-chip SRAM.

Figs. 8 and 9 show the comparison of average execution time for all the GAN inference workloads and static power consumption under these configurations. Compared to a chiplet-based design (Simba), the Chiplet-GAN achieves up to a 63% reduction in execution time as the system scales to 64 * 64 chiplets. Compared to GANAX and SpAtten, the Chiplet-GAN reduces execution time by up to 93% as the system scales. Simba achieves better scalability compared to single-chip designs (i.e., GANAX and SpAtten) due to the chiplet design. However, compared to Chiplet-GAN, Simba still requires more execution time. This is mainly due to the high latency communication between chiplets during inference. In terms of static power consumption, both chiplet systems show an advantage in scalability as the number of components and power consumption linearly increases. Due to the simplicity of GANAX and SpAtten, they consume less power when the system is small (i.e., 4 * 4). However, the power consumption for GANAX and GANPU increases dramatically as the complexity of the global controller must be increased significantly to handle the increase in the computation units. Specifically, compared to Chiplet-GAN, the chiplet-based design (i.e., Simba) consumes up to 10% more power, whereas GANAX and SpAtten consume up to 49% and 59% more static power, respectively, as the system scales.

V. Conclusion

In this article, we analyzed the communication and computation requirements for large-scale GAN model inference and identified three major challenges. Addressing these challenges, we proposed Chiplet-GAN, a chiplet-based accelerator for scaling reduction operation, scaling matrix multiplication, and distributed up-sampling/matrix reshape operations. To the best of our knowledge, this is the first work that proposes a chiplet-based design approach for GANs. The proposed design introduces a novel interconnection fabric with adaptive topology, active/passive network links in NoP, and a workload partition and allocation algorithm. The novel interconnection fabric enables low communication latency during GAN inference. The workload partition and allocation algorithms further reduce communication latency with a greedy algorithm. We conducted extensive simulation studies to demonstrate the effectiveness of Chiplet-GAN. Our detailed evaluation shows that Chiplet-GAN reduces the execution time by 34% and the energy consumption by 21% on average compared to GANAX, SpAtten, and Simba. As the accelerator scales to enhance computation capability for large-scale GAN inference, the proposed design reduces the execution time by up to 63% compared to the existing chiplet-based accelerator (Simba).

Acknowledgment

This work was supported by NSF under Grant CCF-1953961, Grant CCF-1812467, Grant CCF-1812495, Grant CCF-1953980, Grant CCF-1702980, and Grant CCF-1901165.

Yuechen Chen (Member, IEEE), received the Ph.D. degree in computer engineering from George Washington University, Washington, DC, in 2024. He is currently an Assistant Professor with the Department of Computer Science and Information Technologies, Frostburg State University, Maryland, USA. His research interests include parallel computing systems, network-on-chip (NoC), machine learning algorithms, applications, and accelerators.

Ahmed Louri (Fellow, IEEE) received the Ph.D. degree in computer engineering from the University of Southern California, Los Angeles, CA, in 1988. From 1988 to 2015, he was a Professor of electrical and computer engineering at the University of Arizona. From 2010 to 2013, he served as a Program Director of the National Science Foundation's (NSF) Directorate for Computer and Information Science and Engineering. He is currently the David and the Marilyn Karlgaard Endowed Chair Professor of electrical and computer engineering with George Washington University, Washington, DC, USA, which he joined in August 2015. His research interests include interconnection networks and network on chips for multicores, and the use of machine learning techniques for energy-efficient, reliable, high-performance and secure many-core architectures and accelerators. He was the Editor-in-Chief of IEEE Transactions on Computers from 2019 to 2023. In 2024, he is serving on the Computer Society Publication Board Executive Committee. He is also the Chair of the Transactions Operations Committee.

Fabrizio Lombardi (Fellow, IEEE), received the B.Sc. degree (Hons.) in electronic engineering from the University of Essex, U.K., in 1977, the master's degree in microwaves and modern optics and the Diploma degree in microwave engineering from the Microwave Research Unit, University College London, in 1978, and the Ph.D. degree from the University of London in 1982. He is currently the International Test Conference (ITC) Endowed Chair Professorship with Northeastern University, Boston, USA. His research interests include bio-inspired and nano manufacturing/computing, VLSI design, testing, and fault/defect tolerance of digital systems. He was the President of the IEEE Nanotechnology Council in 2022–2023.

Shanshan Liu (Senior Member, IEEE) received the Ph.D. degree in microelectronics and solid-state electronics from the Harbin Institute of Technology, Harbin, China, in 2018. She was a Post-Doctoral Researcher at Northeastern University, Boston, USA, from 2018 to 2021, and an Assistant Professor with New Mexico State University, Las Cruces, from 2021 to 2023. She is currently a Professor with the University of Electronic Science and Technology of China, Chengdu, China. Her research interests include fault tolerance design in high performance computer systems, VLSI design, dependable machine learning, stochastic computing, and error correction codes.

References

[1] T. Karras et al., "Training generative adversarial networks with limited data," in *Advances in Neural Information Processing Systems*. Red Hook, NY, USA: Curran Associates, 2020, pp. 12104–12114.

[2] M. Chen et al., "Generative pretraining from pixels," in *Proc. 37th Int. Conf. Mach. Learn.*, Nov. 2020, pp. 1691–1703.

[3] P. Esser, R. Rombach, and B. Ommer, "Taming transformers for high-resolution image synthesis," presented at the *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2021, pp. 12868–12878.

[4] H. Abdelaziz et al., "Rethinking floating point overheads for mixed precision DNN accelerators," *Proc. Mach. Learn. Syst.*, vol. 3, pp. 223–239, Mar. 2021.

[5] E. Richardson et al., "Encoding in style: A styleGAN encoder for image-to-image translation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2021, pp. 2287–2296.

[6] Y. Jiang, S. Chang, and Z. Wang, "TransGAN: Two pure transformers can make one strong GAN, and that can scale up," in *Advances in Neural Information Processing Systems*. Red Hook, NY, USA: Curran Associates, 2021, pp. 14745–14758.

[7] J.-Y. Zhu et al., "Unpaired image-to-image translation using cycle-consistent adversarial networks," 2020, *arXiv:1703.10593*.

[8] J. Park and Y. Kim, "Styleformer: Transformer based generative adversarial networks with style vector," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2022, pp. 8973–8982.

[9] A. Brock, J. Donahue, and K. Simonyan, "Large scale GAN training for high fidelity natural image synthesis," 2018, *arXiv:1809.11096*.

[10] H. Zhang et al., "Self-attention generative adversarial networks," in *Proc. 36th Int. Conf. Mach. Learn.*, May 2019, pp. 7354–7363.

[11] K. Lee et al., "ViTGAN: Training GANs with vision transformers," 2021, *arXiv:2107.04589*.

[12] A. Yazdanbakhsh et al., "GANAX: A unified MIMD-SIMD acceleration for generative adversarial networks," in *Proc. ACM/IEEE 45th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2018, pp. 650–661.

[13] H. Wang, Z. Zhang, and S. Han, "SpAtten: Efficient sparse attention architecture with cascade token and head pruning," in *Proc. IEEE Int. Symp. High-Perform. Comput. Archit. (HPCA)*, Feb. 2021, pp. 97–110.

[14] S. Kang et al., "GANPU: An energy-efficient multi-DNN training processor for GANs with speculative dual-sparsity exploitation," *IEEE J. Solid-State Circuits*, vol. 56, no. 9, pp. 2845–2857, Sep. 2021.

[15] T. J. Ham et al., "A^3: Accelerating attention mechanisms in neural networks with approximation," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2020, pp. 328–341.

[16] W. Luo et al., "Rethinking motivation of deep neural architectures," *IEEE Circuits Syst. Mag.*, vol. 20, no. 4, pp. 65–76, 4th Quart., 2020, doi: 10.1109/MCAS.2020.3027222.

[17] A. Vaswani et al., "Attention is all you need," 2017, *arXiv:1706.03762*.

[18] Y. S. Shao et al., "Simba: Scaling deep-learning inference with multi-chip-module-based architecture," in *Proc. 52nd Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*. New York, NY, USA: Assoc. Comput. Machinery, Oct. 2019, pp. 14–27.

[19] R. Hwang et al., "Centaur: A chiplet-based, hybrid sparse-dense accelerator for personalized recommendations," in *Proc. ACM/IEEE 47th Annu. Int. Symp. Comput. Archit. (ISCA)*, May 2020, pp. 968–981.

[20] M.-S. Lin et al., "A 7-nm 4-GHz arm[1]-core-based CoWoS[1] chiplet design for high-performance computing," *IEEE J. Solid-State Circuits*, vol. 55, no. 4, pp. 956–966, Apr. 2020.

[21] J.-F. Zhang and Z. Zhang, "Machine learning hardware design for efficiency, flexibility, and scalability," *IEEE Circuits Syst. Mag.*, vol. 23, no. 3, pp. 35–53, 2023.

[22] N. E. Jerger et al., "NoC architectures for silicon interposer systems: Why pay for more wires when you can get them (from your interposer) for free?" in *Proc. 47th Annu. IEEE/ACM Int. Symp. Microarchitecture*, Dec. 2014, pp. 458–470.

[23] A. Kannan, N. E. Jerger, and G. H. Loh, "Enabling interposer-based disintegration of multi-core processors," in *Proc. 48th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Dec. 2015, pp. 546–558.

[24] D. Stow et al., "Cost-effective design of scalable high-performance systems using active and passive interposers," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2017, pp. 728–735.

[25] G. Shan et al., "Architecture of computing system based on chiplet," *Micromachines*, vol. 13, no. 2, p. 205, Jan. 2022.

[26] S. L. Xi et al., "SMAUG: End-to-end full-stack simulation infrastructure for deep learning workloads," *ACM Trans. Archit. Code Optim.*, vol. 17, no. 4, pp. 39:1–39:26, Nov. 2020.