Venus: A Versatile Deep Neural Network Accelerator Architecture Design for Multiple Applications

Jiaqi Yang
Electrical and Computer Engineering
George Washington University
Washington, DC, USA
Yang _ Jiaqi _ Cute@gwu.edu

Hao Zheng
Electrical and Computer Engineering
University of Central Florida
Orlando, Florida, USA
hao.zheng@ucf.edu

Ahmed Louri
Electrical and Computer Engineering
George Washington University
Washington, DC, USA
louri@gwu.edu

***Abstract*—Deep Neural Network (DNN) applications are pervasive. However as demands for these applications continue to increase, so is the challenges for designing flexible and scalable architectures for multi-application implementation. Such accelerators require innovative architecture with flexible Network-on-Chips (NoCs), parallelism exploitation, and better on-chip memory organization to adequately support the diverse computation, memory, and communication needs. In this paper, we propose Venus, a versatile DNN accelerator design that can provide efficient communication and computation support for multi-applications. Venus is a tile-based architecture with a distributed buffer where each tile consists of an array of processing elements (PEs) and a portion of the distributed buffer. The other salient feature of Venus is a flexible Network-on-Chip (NoC) that can dynamically adapt to the communication needs of various running applications thus maximizing data reuse, reducing DRAM accesses, and supporting multiple dataflows with an overall aim of better execution time and better energy efficiency. Simulation results show that our proposed Venus design outperforms state-of-art accelerators (NVDLA [1], ShiDianNao [2], Eyeriss [3], Planaria [4], Simba [5]).**

I. Introduction

Deep Neural Networks (DNNs) have pervaded every aspect of our life such as image recognition, video processing, and many others [6], [7]. Despite their wide popularity, the continued explosion of DNNs is posing many stringent memory bandwidth and communication requirements on the underlying hardware. Even though several DNN accelerators with flexible NoCs [4], [8], [9] have been proposed and implemented, the majority use a centralized buffer design which poses major challenges for running multiple applications at the same time.

The applications of distributed buffers have been proposed to alleviate the bandwidth requirements for DNN applications [5], [10]. For example, Simba [5] has adopted distributed buffers to alleviate the aggregated bandwidth, but unfortunately, it lacks enough flexibility to support various dataflows required for running multiple applications. Multi-application and multi-tenant DNN accelerators with flexible architecture have been proposed [4], [8], [9]. Herald [8] presents a heterogeneous dataflow accelerator, which deploys multiple accelerator substrates (i.e., sub-accelerators), each supporting different dataflow. MAERI [9] proposes a reconfigurable tree-based NoC among the PEs to provide flexible fabric for multiple dataflows. Planaria [4] employed a reconfigurable systolic array, which provides flexible and cost-effective computing resource fission to accommodate different dataflows. However, these accelerators are all based on a centralized buffer architecture which unfortunately can not adequately satisfy the bandwidth, data reuse, and the availability of multiple dataflows required by running multiple applications.

Supporting flexible dataflows with distributed buffers could be challenging, as current dataflows are mostly optimized for accelerators with centralized buffers. Consequently, the direct application [5] of such dataflows may result in data duplication. Data duplication limits the effective capacity of on-chip buffers, resulting in reduced on-chip data reuse opportunities and increased off-chip memory accesses. As a result, data duplication necessitates substantial area overheads for the accelerator chips with increased SRAM buffer size. Therefore, dataflows that can eliminate excessive data duplication and memory access overheads are urgently needed in accelerators with distributed buffers. To the best of our knowledge, very few research efforts have simultaneously proposed a flexible NoC design and distributed buffering with dataflows that can eliminate data duplication and redundant memory accesses in one combined architecture.

In this paper, we introduce Venus, a versatile accelerator architecture with distributed buffers that can accommodate the varied dataflows required by multiple DNNs. The main contributions of this paper are:

- A comprehensive exploration of dataflows and identification of suitable ones that can eliminate excessive data duplication and memory accesses overheads in accelerators with distributed buffers while maintaining the simplicity of on-chip data movement.
- A versatile accelerator architecture to support multiple application execution. The proposed accelerator adopts a tiled architecture design, where each tile consists of a distributed buffer and an array of PEs. The flexible NoC can dynamically be configured to support various traffic patterns of a given dataflow for any given running application.
- Two new algorithms for dataflow selection and hardware configuration. The proposed algorithms can dynamically select suitable dataflow parameters and construct the interconnection configurations, aiming to provide higher performance and energy efficiency.

We evaluate the proposed Venus with a cycle-accurate simulator that can accurately capture the behavior of each hardware component of the DNN accelerator. Simulation results show that our proposed Venus design achieves 81%, 79%, 90%, 75%, and 50% runtime reduction and 73%, 71%, 86%, 69%, and 62% energy consumption reduction on average when compared to baseline designs (NVDLA [1], ShiDianNao [2], Eyeriss [3], Planaria [4], Simba [5]) respectively.

II. Proposed Dataflow

A variety of dataflows have been proposed in current DNN accelerator with centralized buffer for heterogeneous DNN applications execution, but very few of them can be efficiently applied to a design with distributed buffer.

DNNs are composed of a number of convolutional layers, each of which can be formulated with multiple attributes (N, K, C, S, R, X, Y and $X' = X - S + 1, Y' = Y - R + 1$) shown in Fig. 1(a) (stride = 1), to represent the batches of the input activation (N), the number of input channels (C), the number of output channels (K), and the width and height of weight filter (S, R), input activation (X, Y), and output activation (X', Y').

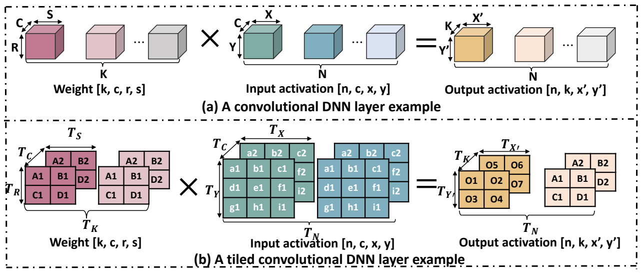
Weight [k, c, r, s]
Input activation [n, c, x, y]
Output activation [n, k, x', y']
(a) A convolutional DNN layer example
Weight [k, c, r, s]
Input activation [n, c, x, y]
Output activation [n, k, x', y']
(b) A tiled convolutional DNN layer example

Fig. 1: (a)A convolutional DNN layer example, (b)a tiled convolutional DNN layer example.

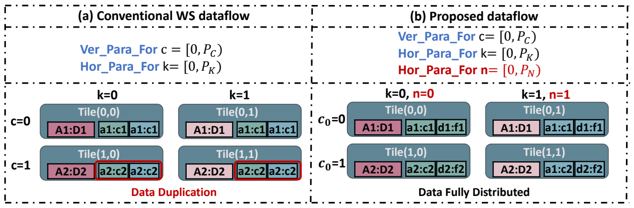
(a) Conventional WS dataflow
Ver_Para_For c = $[0, P_C)$
Hor_Para_For k= $[0, P_K)$
k=0
k=1
c=0
c=1
Tile(0,0)
Tile(0,1)
Tile(1,0)
Tile(1,1)
Data Duplication
(b) Proposed dataflow
Ver_Para_For c= $[0, P_C)$
Hor_Para_For k= $[0, P_K)$
Hor_Para_For n= $[0, P_N)$
k=0, n=0
k=1, n=1
c_0=0
c_0=1
Data Fully Distributed

Fig. 2: (a) Nested loop representation of the parallelization strategy and mapping example of the conventional WS dataflow, (b)Nested loop representation of the parallelization strategy and mapping example of the proposed dataflow.

The high-degree parallelism of loop structures and rhythmic computation patterns are inherently suitable for customized accelerators. Existing DNN accelerators' on-chip buffer capability is limited, each DNN layer has to be tiled into multiple smaller chunks to improve data locality, which is called tiling [11]. Fig. 1(b) shows the tiled convolutional DNN layer example by using the tiling factors $(T_i, i \in \{N, K, C, S, R, X, Y, X', Y'\})$.

Fig. 2(a) shows the nested loop representation of the parallelization strategy and mapping example of the tiled convolutional layer by using conventional weight-stationary (WS) dataflow [1]. The tiled weight and input matrices are distributed into each distributed buffer in parallel, which is called parallelization. The number of partitioned matrices are called parallel factors P_i. The attributes c and k are spatially parallelized (denoted as *Ver_Paral_For* and *Hor_Paral_For*) on a 2×2 titled-based accelerator in horizontal and vertical direction, where each tile represents a collection of computing units and distributed buffer. Since attributes k and c are the indices for the weight matrix ($W_{[k][c][r][s]}$), the weight matrix is fully distributed across all the tiles. On the other hand, the input matrix ($I_{[n][c][x][y]}$) isn't dependent on k, and therefore, the input partition remains duplicating at each row. Because of the data duplication, the on-chip buffer cannot be efficiently utilized. It will diminish possible on-chip data reuse opportunities and increase DRAM accesses with significant energy consumption and latency overhead. The major issue of conventional dataflows is the data duplication or complicated communication patterns when they are applied to a design with a distributed buffer. This raises the need for the dataflows that can efficiently be applied to a DNN accelerator with distributed buffer.

To meet the raised requirements, we need to examine the root cause of data duplication in existing designs. The primary reason is that only one index (c) of the input matrix ($I_{[n][c][x][y]}$) is parallelized in a two dimensional accelerator design. As such, to fully distribute a matrix, at least two indexes have to be parallelized across each row and column. For example, we use a parallelization strategy depicted in Fig. 2(b) to illustrate the concept. For a nested loop with multiple attributes, both weight and input matrices can be partitioned and distributed in 2 × 2 tiles. Each tile consists of a distributed buffer

	Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Parallelization Strategies	C											✓								
	N	✓	✓	✓							✓	✓	✓	✓		✓	✓			✓
	K	✓			✓	✓				✓	✓	✓	✓			✓		✓		✓
	S			✓			✓	✓		✓		✓	✓	✓			✓		✓	✓
	R		✓						✓		✓			✓	✓	✓	✓	✓	✓	✓
	X				✓		✓		✓	✓	✓	✓		✓				✓	✓	✓
	Y					✓		✓		✓			✓		✓	✓	✓	✓	✓	✓
Supported dataflows		IS/WS													IS/WS/RS					IS/WS/RS/OS

Fig. 3: All dataflow candidates that are suited for distributed buffer.

to store all the input, weight, and output matrices. If attributes c and n are parallelized in different directions (Horizontal and Vertical), the input matrix will be fully distributed across each row and column. Similarly, the weight matrix will be fully distributed. This is because c and k are two attributes of the weight matrix. Then the on-chip buffer can be fully utilized and the proposed dataflow can effectively reduce the off-chip memory accesses by increasing the data reuse opportunities.

Following this principle, many parallelization strategies are qualified to avoid the data duplication issue. The rationale behind our choice is to reduce the communication complexity. Unlike conventional CPUs with sophisticated hardware support, it is relatively hard to enable fine-grained data movement in DNN accelerators with scratchpad memory, where data movement is managed by software. Given this, we would like to execute DNN layers with simple data movement patterns like ring. To achieve this, we observe that both input and weight matrices have to be partitioned with the equivalent number of indexes. In addition, to reduce the diameter of ring, attribute c has to be parallelized, as each channel is independent of each others. In other words, there is no communication between input and weight matrices across different channels.

As a result, the dataflows that can efficiently be applied to a design with a distributed buffer have three requirements: (1) at least two indexes of both weight and input matrices have to be parallelized across each row and column, (2) both weight and input matrices have to be partitioned with the equivalent number of indexes, and (3) atrribute c must be parallelized. All the supported dataflow candidates are shown in Fig. 3.

III. Venus Architecture

The proposed overall Venus accelerator design aims to provide adequate computation and communication support in pursuit of various data reuse opportunities of heterogeneous DNN applications. To achieve this, Venus adopts a tiled architecture design to improve overall on-chip memory bandwidth. In addition, a flexible NoC is designed to enable the flexible dataflow choices for heterogeneous DNN applications execution.

The overall Venus consists of a control unit, interconnects, and a collection of computation units. The control unit connects to the host (e.g., CPUs) via a host interface. The control unit receives requests from the host and stores them in the request dispatcher, shown in Fig. 4. The request dispatcher sends the compiled request to the dataflow selection unit to select the suitable dataflow. Then the hardware configuration unit creates the interconnect configuration that can enable the execution of the selected dataflow according to the result of the dataflow selection unit.

The accelerator uses configuration instructions to configure the interconnects so that it can flexibly enable on-chip communication and data reuse of heterogeneous DNN layers. The accelerator also uses dataflow instructions to retrieve and process data from memory. The instruction dispatcher, shown in Fig. 4, features a controller

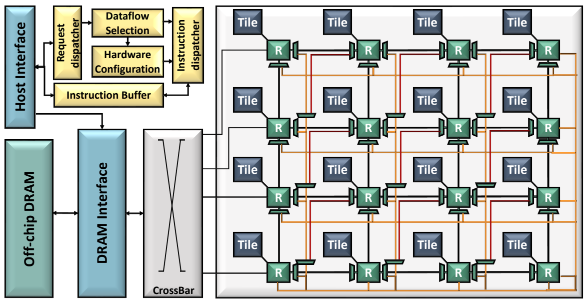

Fig. 4: Venus accelerator architecture(an example of 4 × 4 architecture).

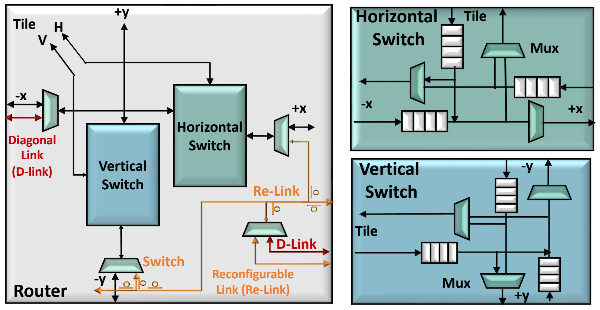

Fig. 6: (a) Proposed router architecture, and (b) vertical and horizontal switches.

which keeps track of instruction issues and completion. The controller generates addresses for the instruction buffer, which forwards instructions to the decoder unit. The DRAM is connected to the proposed accelerator through a DRAM interface. To increase the endpoint bandwidth at the DRAM interface, a crossbar is implemented to support all-to-all communication. The accelerator adopts a tiled-based design, where 4×4 tiles are connected via the flexible interconnects, shown in Fig. 4.

A. Proposed Tile and PE Architecture

Each tile consists of a distributed buffer, a router interface, a 4 × 4 array of processing elements (PEs), and a reuse First-in-First-Out (FIFO) buffer, shown in Fig. 5. The tile connects to the router via a router interface. The 4 × 4 processing elements are connected by the mesh topology. To support efficient data reuse, we allow inter-tile data reuse on the tile interconnect, where each tile can send locally-stored data to other tiles. The reuse FIFO acts as a double buffer [12], which can support inter-PE communication. This is designed to enable the data exchange between distributed buffers, reducing off-chip memory accesses. Each PE consists of a local buffer, a data dispatcher, a MAC array, and processing units (e.g., ReLu).

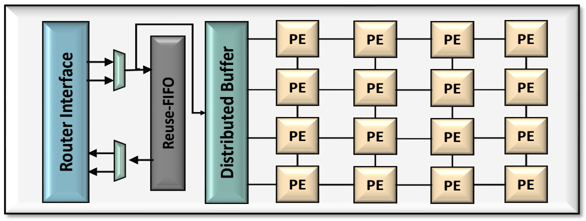

Fig. 5: Proposed tile architecture.

B. Flexible NoC Design

The proposed NoC Design aims to support various communication patterns by supporting various dataflows. Even though previous work [13] has addressed the communication patterns between off-chip memory and global buffers in various dataflows, the communication support between tiles remains unexplored, in particular for distributed buffers. To solve this, we explore and propose a flexible NoC design.

In general, the flexible NoC design enables data propagation between adjacent tiles. This can reduce the complexity of data exchange and avoid complicated communication protocols. Specifically, the proposed NoC design can be partitioned into multiple ring topologies with any size and location. Those rings will be formed to propagate the weights and input activations. The proposed NoC design also provides diagonal communication besides normal vertical and horizontal communication. The proposed flexible NoC consists of flexible routers, reconfigurable links, and diagonal links, which will be further discussed.

Flexible Router : The major functionality required for ring topology is forward and eject/inject in-flight packets. This significantly simplifies the router design with much reduced radix. As shown in Fig. 6, the flexible router consists of a vertical switch and a horizontal switch. The vertical switch processes the column-wise communication, and the horizontal switch processes the row-wise communication. Vertical switches are connected to reconfigurable links (Re-link), and horizontal switches are connected to Re-link and diagonal links (D-link). The Re-link consists of multiple simple transistors to turn on/off the bypassing link connection between two routers, avoiding signal interference. The Re-link also connects the vertical and horizontal switches in the same router together to fully utilize the radix. D-link is used to connect a pair of routers sitting across the diagonal, which can effectively reduce the communication distance and hop count and bridge non-adjacent routers.

IV. Dataflow Selection

The primary goal of dataflow selection is to reduce the off-chip memory accesses. The off-chip memory access is determined by two factors: data volume of each invocation and the number of invocations. To better illustrate the problem, we use an example depicted in Fig. 2.

Recall that DNN layer is represented with multiple attributes ($i, i \in \{N, K, C, S, R, X, Y, X', Y'\}$ and $X' = X - S + 1, Y' = Y - R + 1$). After the loop tiling, the tiled weight and input matrices have to be distributed into multiple tiles. The number of partitioned matrices is called parallel factors (P_i). And, the data volume in each partitioned matrix is represented as i_1.

To estimate the DRAM access volume (DA) for the dataflow, we calculate the product of the data volume involved in each invocation (V_d) and the total number of invocations (R_d) for all data types (i.e., weight(wt), input activation($ifmap$), psum($psum$)) as shown in Algorithm 1 (Line 7). The data volume involved in each invocation for different data type can be calculated as V_{wt}, V_{ifmap}, V_{psum} as depicted in Equation 1:

$$\begin{aligned} V_{wt} &= \prod i_1 \times P_i, i \in \{K, C, S, R\} \\ V_{ifmap} &= \prod i_1 \times P_i, i \in \{N, C, X, Y\} \\ V_{psum} &= \prod i_1 \times P_i, i \in \{N, K, X', Y'\} \end{aligned} \tag{1}$$

The parallel factors determine the accessed array regions of each partitioned matrix, thus different parallel factor configurations imply different dataflows. We observed that different dataflows exhibit large differences in DRAM access volume. The total number of invocations can be calculated as R_{wt}, R_{ifmap}, R_{psum}. In this section, we use weight stationary dataflow as an example to explain the equations but it also can support output/input/row stationary dataflow. The total

Algorithm 1: Dataflow Parallelism Optimization

```
   Input  : Possible fully partitioned and multi-dimension parallel dataflow
            candidates $DF$
   Input  : Dimensions of the DNN layer of each DNN layer
            $i^m, i \in \{K, C, R, S, N, X, Y\}, m \in [0, M)$
   Input  : Dimensions of the a sub-layer in distributed buffer
            $i_1^m, i \in \{K, C, R, S, N, X, Y\}, m \in [0, M)$
   Input  : Distributed buffer capacity $C_{DB}$
   Output: $Parallelfactor : P_i^m, i \in \{K, C, R, S, N, X, Y\}, m \in [0, M)$
   Output: $OptimalDataflow$
 1 for $m \in [0, M)$ do
 2     for $P_i^m \in DF$ do
 3         // Data volume involved in each invocation
 4         Calculate $V_d^m, d \in \{wt, ifmap, psum\}, m \in [0, M)$;
 5         // Number of invocations
 6         Calculate $R_d^m, d \in \{wt, ifmap, psum\}, m \in [0, M)$;
 7         // Number of invocations
 8         $DA^m = \sum_d V_d \times R_d, d \in \{wt, ifmap, psum\}, m \in [0, M)$;
 9         for $datavolume^m \in [0, C_{DB}]$ do
10             $Find\ the\ Minimal\ DA^m$;
11         end
12         return $P_i^m$;
13     end
14     return *Optimal Dataflow*
15 end
```

number of invocations (R_d) of weight stationary dataflow can be calculated as depicted in Equation 2:

$$\begin{aligned}
R_{wt} &= \prod \frac{i}{i_1 \times P_i}, i \in \{K, C, S, R\} \\
R_{ifmap} &= \prod \frac{i}{i_1 \times P_i}, i \in \{N, K, C, S, R, X', Y'\} \\
R_{psum} &= \frac{\prod^{i\in\{N,K,S,R,X',Y'\}} i \times (\frac{2C}{C_1 \times P_C} - 1)}{\prod^{i\in\{N,K,S,R,X',Y'\}} i_1 \times P_i}
\end{aligned} \tag{2}$$

The data volume of the partitioned matrices ($datavolume$) can be calculated as Equation 3:

$$\begin{aligned}
datavolume = &\prod i_1, i \in \{K, C, S, R\} + \\
&\prod i_1, i \in \{N, C, X, Y\} + \\
&\prod i_1, i \in \{N, C, X', Y'\}
\end{aligned} \tag{3}$$

The data volume of the partitioned matrices is limited by the distributed buffer capacity(C_{DB}) shown in Algorithm 1 (Line 8). Then we can find the parallel factors that can minimize the DRAM accesses with the distributed buffer capacity(C_{DB}) limitation.

We compare the total DRAM accesses volume (DA) of all candidate dataflows(DF) and consider the dataflow with minimal DRAM accesses volume as the optimal one for each DNN layer. The dataflow selection algorithm is described in detail in Algorithm 1.

V. Dynamic Hardware Automation

This section will go through the hardware configuration required to allow flexible dataflow for DNN operation. As previously stated, the parallelization strategy and loop order are determined by the dataflow selection algorithm. Following the choice, the proposed design will be dynamically configured into several topologies that are suited for optimal dataflow.

The NoC of accelerator will be configured to support the communication patterns of the selected dataflow. For example, as shown in Fig. 7, multiple rings are configured to support various communication patterns of different dataflows. To better illustrate the proposed design, we provide three examples to represent three representative dataflows: weight-stationary, row-stationary, and output-stationary. The difference among these dataflows is the data movement patterns of input, weight, and partial sums. For example, in Fig. 7, a weight stationary dataflow is deployed, where P_C = 2, P_N = 2, P_K = 4, P_S = 2, and P_X =4. As attribute C is divided into two parts, two rings will be generated. In each ring, the input data is forwarded, while partial sums are accumulated horizontally.

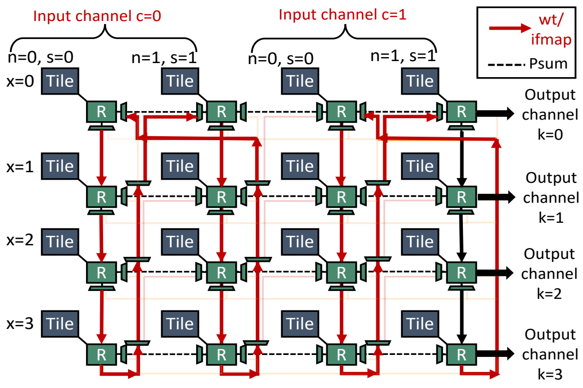

Fig. 7: Topology configuration of weight stationary dataflow with parallelization strategies ($P_C = P_N = P_S = 2, P_K = P_X = 4$).

As the data reuse of weights and input activations only exist in the same input channel C, all communications involving weights and input activations should be restricted to the same input channel C and other constraints should be determined according to different data movement paths (communication patterns between tiles) of various dataflows. As a result, we can summarize that each partition with the same attribute C will be clustered into the same ring in weight/input stationary dataflow to transfer input activations or weights. Each partition with the same attributes C and Y will be clustered into the same ring in row stationary dataflow to forward input activations. In output stationary dataflow, each partition with the same attributes C, X, and Y will be clustered into the same ring to transfer input activations and each partition with the same attributes C, S, and R will be clustered into the same ring to forward weights. Within each ring, each tile will forward the data to its bottom tile given a simpler routing algorithm. This will naturally avoid the deadlock while maintaining the simplicity of the routing.

VI. Evaluation

A. Simulation Setup

In this section, we discuss the methodologies to compare our proposed accelerator with other state-of-the-art baselines [1]–[5] in terms of performance, DRAM access, energy consumption, and area consumption.

Simulator: We utilize a customized version of the open-source Timeloop simulator [15]. We extend this simulator to support the non-uniform distribution of latency and bandwidth between PEs and tiles. In order to obtain execution time results, the simulator monitors the number of arithmetic operations and the number of accesses to each memory hierarchy, taking the dataflow and system configuration parameters into account. The number of arithmetic operations is used to calculate the computation time, while the number of accesses to each on-package memory hierarchy is used to calculate the on-package communication time. The off-package communication time is obtained from the DRAMSim2 simulator [16]. The overall execution time is derived by adding up the computation time, the on-package communication time, and the off-package communication time, considering the overlap caused by the buffering of the distributed buffer and other memory hierarchies.

Accelerator Modeling : We implement the proposed design including 32 $\times$ 32 tiles interconnected by a flexible NoC. Each tile consists of 4 $\times$ 4 processing elements (PEs), a distributed buffer,

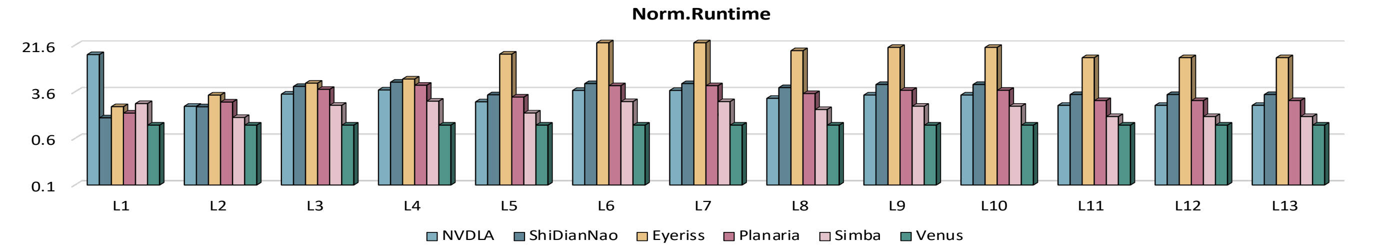

Fig. 8: Normalized runtime for each convolutional layer of DNN model VGG-16 [14] by using baselines and our proposed accelerator, normalized to the runtime of proposed accelerator.

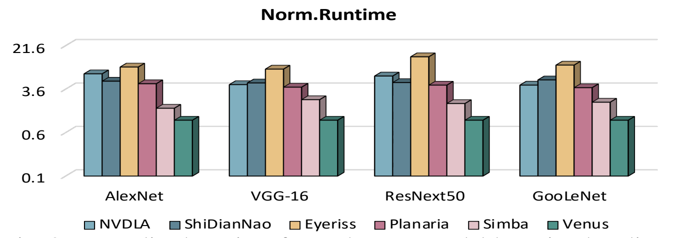

Fig. 9: Normalized runtime for each DNN model by using baselines and our proposed accelerator),normalized to the runtime of proposed accelerator.

and a FIFO buffer. Each PE consists of local buffer, data dispatcher, MAC array, and processing unit (e.g.,ReLu). The on-chip frequency of the proposed accelerator is 700MHz. The on-chip distributed buffer capacity of each tile is 100KB. For a fair comparison, we keep the configurations consistent for all baseline designs (NVDLA [1], ShiDianNao [2], Eyeriss [3], Planaria [4], Simba [5], and our proposed accelerator): All designs use 16384 processing elements, and each processing element contains 16 MAC units, the SRAM of each processing element is 5KB and the total on-chip SRAM capacity is 100MB.

Dataflow Modeling: For our evaluations, we use our proposed dataflow for the proposed accelerator. The proposed dataflow has optimal parallelization strategy and loop order created by Algorithm 1 according to different DNN layers' requirements. We use conventional weight stationary dataflow for NVDLA [1] and Simba [5], row stationary dataflow for Eyeriss [3], output stationary dataflow for ShiDianNao [2], all dataflows for Planaria [4].

B. *Performance Analysis*

Fig. 8 shows the normalized runtime for each convolutional layer of DNN model VGG-16 [14] by using different acceleration platforms. The computation time of all the designs is very close because the amount of multiplication and accumulation computations (MACs) of each DNN layer is the same despite being performed in different architectures. The on-package communication of the proposed design is larger than the baselines because the proposed design uses routers to ensure the communication between tiles but Planaria, NVDLA, ShiDianNao, and Eyeriss only use simple interconnects (buses and mesh links) to enable the communication between PEs. The proposed design has more communication between tiles than Simba but it reduces the average hop latency by using simple routers. Simba and the proposed accelerator have lower off-package memory bandwidth requirements than other baselines because of the scalable architecture. Planaria [4] can reduce the DRAM accesses by supporting multiple dataflows with reconfigurable interconnects but it still suffers from bandwidth limitation which will hurt the performance. The main reason our design outperforms other accelerators is that our design has reduced memory bandwidth requirement, reduced DRAM accesses and fully utilized hardware resources which dominate the overall runtime. The fixed parallelization strategy of the compared designs limits their ability to exploit intra-layer data reuse and hardware resources to minimize DRAM accesses. The DRAM access analyses are shown in the following part. The proposed accelerator achieves 72%, 74%, 90%, 70%, and 48% runtime reduction on average for convolutional layers of VGG-16 [14], when compared to baselines respectively.

Fig. 9 shows the normalized runtime of different DNN models including AlexNet [17], VGG-16 [14], ResNext50 [18], and GoogLeNet [6] by using different acceleration platforms. As can be seen, our proposed design outperforms other accelerators model-wise. The proposed accelerator achieves 81%, 79%, 90%, 75%, and 50% runtime reduction on average for DNN models, when compared to baselines respectively.

C. *DRAM Access Analysis*

Fig. 10 shows the normalized DRAM accesses of baselines and proposed accelerator in each convolutional layer of DNN model VGG-16 [14]. As can be seen, the proposed accelerator outperforms the baselines layer-wise, because the conventional dataflow can't fully distribute data and can only use fixed parallel policy and tiling factors. This limits their ability to minimize DRAM accesses and fully utilize hardware resources including the buffer capacity, computation resources. What's more, the proposed dataflow can fully distribute data to avoid data duplication and also can support multiple parallelization strategies so that it is flexible enough to fully utilize the hardware resources and minimize the DRAM accesses with the Algorithm 1. For example, the first layer of VGG-16 has few input channels so NVDLA and Simba that use weight stationary dataflow to parallelize input channels will have low performance. NVDLA is worse than Simba because NVDLA has higher memory bandwidth requirements which lead to less parallelism. Planaria [4] can reduce the DRAM accesses through the flexibility of dataflows but it doesn't have the dataflow selection algorithm to select the optimal dataflow for different DNN layers. The proposed design achieves 57%, 69%, 90%, 58% and 60% DRAM accesses reduction on average for convolutional layers of VGG-16 [14] when compared to baselines respectively. Fig. 11 shows the normalized DRAM accesses of baselines and proposed designs in each DNN model. The proposed design achieves 75%, 72%, 84%, 70%, and 57% DRAM accesses reduction on average for DNN models, when compared to baselines respectively.

D. *Energy Consumption Analysis*

For energy analysis, we use DSENT [19] to obtain power consumption and Timeloop [20] to calculate runtime. It should be noted that the evaluation includes the energy consumption of the entire system including control units, computation units, DRAM, distributed buffer, local buffers, and interconnects. Fig. 12 shows the normalized overall energy consumption analysis of the proposed accelerator. As can be seen, the proposed accelerator achieves 73%, 71%, 86%, 69% and

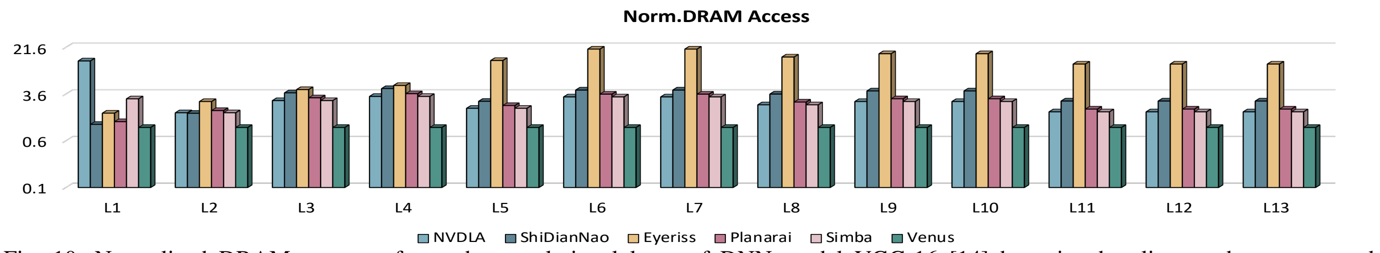

Fig. 10: Normalized DRAM accesses for each convolutional layer of DNN model VGG-16 [14] by using baselines and our proposed accelerator), normalized to the DRAM accesses of proposed accelerator.

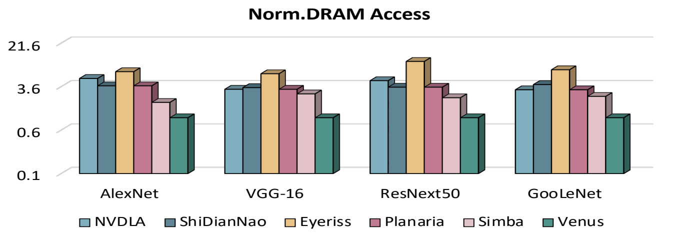

Fig. 11: Normalized DRAM accesses for each DNN model by baselines and our proposed accelerator),normalized to the DRAM accesses of proposed accelerator.

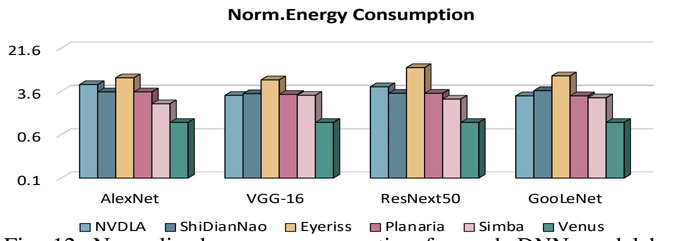

Fig. 12: Normalized energy consumption for each DNN model by using baselines and the proposed accelerator, normalized to the energy consumption of proposed accelerator.

62% energy consumption reduction on average for DNN models, when compared to baselines. The main reasons for the reduction are reduced DRAM accesses (benefit from the proposed dataflow and the dataflow selection algorithm 1), simple interconnects, reduced long-distance communication(benefit from the multiple dimensions parallelization strategy), and low configuration time (benefit from the Hardware Automation).

E. Area Analysis

We evaluate the area consumption of the various architectures under TSMC 40 nm technology, the MAC array consumes only 7.1% of the total PE area, while the memory hierarchy, SMB, and IDMB/ODMB, consumes a majority of the total area, 82.9%. The PE control unit consumes 3.7% of the total PE area. The total area consumption takes PE, SRAM, flexible interconnects, and control logic into account. For the entire proposed accelerator, the PE array, which consists of 16384 PEs consumes 65.23% of the overall chip area. The controller consumes 0.6% total chip area which is negligible. The additional components for the flexible interconnects including flexible routers, reconfigurable links, diagonal links, and muxes consume 4.0% of the total chip area.

VII. Conclusion

In this paper, we propose Venus, a versatile accelerator architecture with distributed buffers that accommodate the varied dataflows required by heterogeneous DNNs based on the communication and computing needs of multiple DNN applications. The proposed Venus adopts tiled architecture, where each tile consists of an array of processing elements (PEs) and a distributed buffer. The tiles are interconnected by a flexible Network-on-Chip (NoC). Specifically, the proposed design can be configured to support various communication patterns that enable the flexible dataflow choices for heterogeneous DNN applications execution, thus providing desired performance and energy efficiency. Simulation shows that our proposed Venus design achieves 81%, 79%, 90%, 75%, and 50% runtime reduction and 73%, 71%, 86%, 69%, and 62% energy consumption reduction on average when compared to baseline designs respectively.

References

[1] Nvdla deep learning accelerator, http://nvdla.org, 2017.

[2] Zidong Du et al. Shidiannao: Shifting vision processing closer to the sensor. In *Proc.of ISCA*. IEEE, 2015.

[3] Yu-Hsin Chen et al. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *Solid-state circuits*, 2016.

[4] Soroush Ghodrati et al. Planaria: Dynamic architecture fission for spatial multi-tenant acceleration of deep neural networks. In *Proc.of MICRO*. IEEE, 2020.

[5] Yakun Sophia Shao et al. Simba: Scaling deep-learning inference with multi-chip-module-based architecture. In *Proc.of MICRO*. IEEE, 2019.

[6] Christian Szegedy et al. Going deeper with convolutions. In *Proc.of CVPR*. IEEE, 2015.

[7] Subhashini Venugopalan et al. Sequence to sequence-video to text. In *Proc.of ICCV*, 2015.

[8] Hyoukjun Kwon et al. Heterogeneous dataflow accelerators for multi-dnn workloads. In *Proc.of the ISCA*. IEEE, 2021.

[9] Hyoukjun Kwon et al. Maeri: Enabling flexible dataflow mapping over dnn accelerators via reconfigurable interconnects. *ACM SIGPLAN Notices*.

[10] Mohit Upadhyay et al. React: a heterogeneous reconfigurable neural network accelerator with software-configurable nocs for training and inference on wearables. In *Proc.of DAC*, 2022.

[11] Hyoukjun Kwon et al. Understanding reuse, performance, and hardware cost of dnn dataflow: A data-centric approach. In *Proc.of MICRO*. IEEE, 2019.

[12] Chen Zhang et al. Optimizing fpga-based accelerator design for deep convolutional neural networks. In *Proc.of FPGA*. ACM/SIGDA, 2015.

[13] Jiaqi Yang et al. Adapt-flow: A flexible dnn accelerator architecture for heterogeneous dataflow implementation. In *Proc.of GLSVLSI*. ACM, 2022.

[14] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *Proc.of ICLR*. IEEE, 2014.

[15] Angshuman Parashar et al. Timeloop: A systematic approach to dnn accelerator evaluation. In *Proc.of ISPASS*. IEEE, 2019.

[16] Paul Rosenfeld et al. Dramsim2: A cycle accurate memory system simulator. *CAL*, 2011.

[17] Alex Krizhevsky et al. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 2017.

[18] Saining Xie et al. Aggregated residual transformations for deep neural networks. In *Proc.of CVPR*. IEEE, 2017.

[19] Chen Sun et al. Dsent-a tool connecting emerging photonics with electronics for opto-electronic networks-on-chip modeling. In *Proc.of NOCS*. IEEE, 2012.

[20] Sheng-Chun Kao and Tushar Krishna. Magma: An optimization framework for mapping multiple dnns on multiple accelerator cores. In *Proc.of HPCA*, 2022.