Versa-DNN: A Versatile Architecture Enabling High-Performance and Energy-Efficient Multi-DNN Acceleration

Jiaqi Yang, *Member, IEEE*, Hao Zheng, *Member, IEEE*, and Ahmed Louri, *Fellow, IEEE*

***Abstract*—Emerging applications utilize numerous Deep Neural Networks (DNNs) to address multiple tasks simultaneously. As these applications continue to expand, there is a growing need for off-chip memory access optimization and innovative architectures that can adapt to diverse computation, memory, and communication requirements of various DNN models. To address these challenges, we propose Versa-DNN, a versatile DNN accelerator that can provide efficient computation, memory, and communication support for the simultaneous execution of multiple DNNs. Versa-DNN features three unique designs: a flexible off-chip memory access optimization strategy, adaptable communication fabrics, and a communication and computational aware scheduling algorithm. The proposed off-chip memory optimization strategy can improve performance and energy efficiency by increasing hardware utilization, eliminating excess data duplication, and reducing off-chip memory accesses. The adaptable communication fabrics consist of distributed buffers, processing elements, and a flexible Network-on-Chip (NoC), which can dynamically morph and fission to support distinct communication and computation needs for simultaneously running DNN models. Furthermore, the proposed scheduling policy manages the simultaneous execution of multiple DNN models with improved performance and energy efficiency. Simulation results using several DNN models, show that the proposed Versa-DNN architecture achieves 41%, 238%, 392% throughput speedup and 30%, 59%, 63% energy reduction on average for different workloads when compared to state-of-the-art accelerators such as Planaria, Herald, and AI-MT, respectively.**

***Index Terms*—Deep neural networks (DNNs), multi-DNN, dataflow accelerators, domain-specific accelerators, memory access optimization, Network-on-Chip(NoC).**

I. Introduction

DEEP Neural Networks (DNNs) have emerged as the backbone of numerous artificial intelligence applications to infer complex decisions. Specialized hardware accelerators, such as GPUs and TPUs, have been developed to expedite the training and inference of DNNs. However, there has been a surge in applications that require the concurrent execution of multiple complex functionalities, such as vision-centric autonomous systems and Cloud-based systems [1], [2]. For example, today's computer vision tasks, such as object detection and semantic segmentation, involve multiple DNN models to achieve accurate perception. Similarly, a set of DNNs are combined to enhance the quality of camera-captured content or provide advanced augmented reality features [3].

Despite the surge of multi-DNN models, it remains a challenge to facilitate the multi-DNN execution due to distinct computation and communication characteristics. For example, DNN models often involve varying operations and layer sizes, all with unique computation and communication ratios and memory access patterns. Current DNN accelerators are mostly optimized for a given set of DNN models, and thus they are inefficient in handling various computation and communication characteristics. For example, the datapath of Eyeriss [4] is optimized to handle row-stationary dataflow, and it has limited applicability to support other dataflows. The problem is further exacerbated by resource contention, as network-on-chip, computation units, and memory are shared among running DNN models.

Significant research efforts [1], [3], [5], [6], [7] have been dedicated to enhancing the performance and energy efficiency of simultaneously accelerating multiple DNNs on the same accelerator. However, current multi-DNN accelerators have not yet addressed all the challenges. For instance, Herald [1] provides multiple accelerator options with different memory access optimization techniques, but the computing and memory capabilities are determined at the design stage, making it challenging to adapt to the dynamic demands and interactions of running DNNs. Similarly, AI-MT [5] adopts a systolic array architecture that is not efficient at handling a variety of off-chip memory access optimization strategies, as its data path only supports a specific type of data movement.

Planaria [6] proposes a flexible accelerator architecture that can be dynamically partitioned to accommodate several DNN models, in which compute and memory resources are dynamically determined and allocated. While the dynamic resource allocation can alleviate the resource underutilization issue, its rigid dataflow design still prevents the further optimization such as reduced off-chip memory access. Moreover, like Herald [1], MAGMA [8] and Veltair [9], Planaria [6]'s scheduling policy only considers compute resources and neglects the impacts

Manuscript received 11 May 2023; revised 29 October 2023; accepted 4 December 2023. Date of publication 8 December 2023; date of current version 10 January 2024. This work was supported in part by the National Science Foundation under Grants CCF-1901165, CCF-1702980, CCF-1953980, and CCF-2131946. Recommended for acceptance by Y. Yang. *(Corresponding author: Jiaqi Yang.)*

Jiaqi Yang and Ahmed Louri are with the Department of Electrical and Computer Engineering, George Washington University, Washington, DC 20052 USA (e-mail: yang_jiaqi_cute@gwu.edu; louri@gwu.edu).

Hao Zheng is with the Department of Electrical and Computer Engineering, University of Central Florida, Orlando, FL 32816 USA (e-mail: hao.zheng@ucf.edu).

Digital Object Identifier 10.1109/TPDS.2023.3340953

of communication latency. On the other hand, AI-MT [5], PREMA [7], and LayerWeaver [10] adopt a time-multiplexing approach to interleave the execution of multiple DNN models based on their computational impacts. Such scheduling policies are unlikely to be generalized to handle the spatial resource management.

There is a pressing need for further research to improve the performance and energy efficiency of multi-DNN accelerators. To this end, we propose Versa-DNN, a versatile DNN accelerator that can provide efficient computation, memory, and communication support for the simultaneous execution of multiple DNNs. The main contributions of this paper are:

- We propose an adaptable accelerator fabric consisting of distributed buffers, processing elements, and a flexible Network-on-Chip (NoC), which can dynamically morph and fission to simultaneously support distinct communication needs for running DNN models.
- We explore various parallelism strategies for multi-DNN accelerators compatible with distributed buffer configurations. The proposed off-chip memory access optimization strategy selection algorithm can select suitable parameters for parallel DNN execution to avoid data duplication, thus trading off expensive and energy-inefficient off-chip memory access for energy-efficient on-chip data movement. More importantly, the proposed strategy maintains the simplicity of on-chip data movement with hop-to-hop communications.
- We propose a communication and computational aware scheduling algorithm for accelerator partition and hardware configuration. The proposed algorithms can dynamically select suitable accelerator partitioning strategies and interconnection configurations with the aim of satisfying both computation and communication resource constraints.

We evaluate the proposed Versa-DNN with a cycle-accurate simulator that can accurately capture the behavior of each hardware component of the DNN accelerator. Our evaluation uses the same benchmark DNN models as compared to state-of-the-art accelerators, and the simulation results demonstrate that our proposed Versa-DNN architecture achieves 41%, 238%, 392% throughput speedup and 30%, 59%, 63% energy reduction on average for different workloads when compared to Planaria [6], Herald [1], and AI-MT [5], respectively. We believe that the proposed Versa-DNN accelerator provides a significant addition to the design of high-performance and energy-efficient multi-DNN accelerators.

II. Background

A. DNNs Workload

Deep Neural Networks (DNNs) have been widely used in many applications, such as image recognition, speech recognition, and natural language processing. These networks are composed of a number of convolutional layers, each of which can be formulated as a nested loop with multiple attributes (e.g., N, K, C, R, S, Y, X), as shown in Fig. 1 (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'=X-S+1, Y'=Y-R+1). The operation of each convolutional layer is primarily composed of high-dimensional convolutions as shown in Fig. 1. In this computation, input activation is convolved with the weight matrix in the same input channel. The results of the convolution at each point are summed across all the input channels. The result of this computation is the output activation that comprises one channel of output feature map (ofmap). The weight can be used on the same input to create additional output channels. As such, multiple input feature maps could be processed together as a batch to potentially improve the reuse of the weights. The high-degree parallelism of loop structures and rhythmic computation patterns are inherently suitable for customized accelerators.

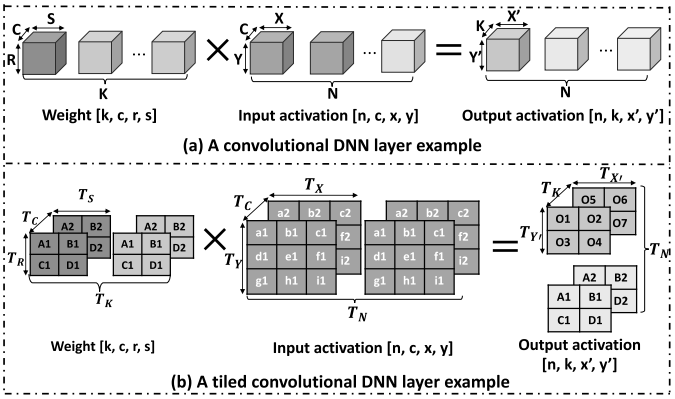

Fig. 1. (a) Convolutional DNN layer example, (b) tiled convolutional DNN layer example.

B. DNNs Dataflows

DNN dataflow refers to the parallelism strategies that can efficiently exploit both spatial and temporal data reuse. In general, there are several types of commonly used dataflows, including weight-stationary, input-stationary, output-stationary, and row-stationary dataflows.

Weight-Stationary Dataflow (WS): In weight-stationary dataflow, the weights are preloaded to PE array, and the input data is streamed through the network. In this dataflow, the loop order for computations is arranged to ensure that the weight values, K, C, R or S, are placed in the outermost loop. As a result, the input activations are in the inner loops. This allows the same weight values to be reused multiple times for different input activations, improving memory access efficiency. The WS dataflow can benefit large convolutional neural networks where the weight matrices can be very large and expensive to store and access.

Input-Stationary Dataflow (IS): In input-stationary dataflow, the input data are stored in local register, and the weight is streamed through the network. In this dataflow, the loop order for computations is arranged in a way that the input data, N, C, X or Y, are accessed in the outermost loop, followed by the weight in the inner loops. This allows the same input activations to be

reused multiple times for different weight, improving memory access efficiency. As such, it can significantly reduce the data movement when the input matrix is large.

Output-Stationary Dataflow (OS): Like WS and IS dataflows, output-stationary dataflow is designed to retain output matrix in the local register for partial sum accumulation. Consequently, the weights and inputs flow through PE array to produce partial results. The data movement can be significantly reduced when partial row accumulation happens frequently.

Row-Stationary Dataflow (RS): RS dataflow is an alternative means to explore the data reuse, in which input, weight, and output data are temporarily reused at row-wise. N, K, X', or Y' dimensions are arranged in the outermost loop, and it is beneficial for applications with large input and output data.

Overall, the choice of dataflow mainly depends on the specific requirements of the application and the available hardware resources. Given the high dimensionality of DNNs, dataflow is a critical factor for off-chip memory access and computation performance.

III. Proposed Versa-DNN Design

DNN models are composed of different operations and layers, each with their own computation and communication ratio and memory access patterns. This necessitates the use of contemporary DNN accelerators that are capable of managing various computation and communication characteristics at the same time. In addition, resource contention is another major concern that limits the accelerator performance. The goal of our proposed design is to simultaneously address the computation and communication challenges when running multiple DNNs with distinct characteristics. To achieve this, we propose a novel framework to support the simultaneous execution of DNNs spanning architecture, dataflow, and scheduling. Specifically, we propose three unique designs: (1) universal and modular memory and communication fabrics to support dynamic architecture partitioning, (2) a flexible dataflow to pursue both communication simplicity and dataflow flexibility, and (3) a scheduling algorithm to consider the effects of communication and computation. The details of the proposed architecture and dataflow will be discussed in the following section.

A. Overall Versa-DNN Architecture Design

As shown in Fig. 2, the overall Versa-DNN architecture comprises a control unit and multiple computation units. The control unit is connected to the host (e.g., CPU) through a host interface. The control unit receives requests from the host and stores them in the request dispatcher. The proposed design implements instructions necessary for popular inference services (e.g., CNN), including convolution, vector-vector operations, activation, batch normalization, and pooling. The accelerator also uses instructions to configure the interconnects so that it can flexibly enable on-chip communication. The instruction dispatcher, shown in Fig. 2, features a controller 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. A crossbar is implemented to increase the endpoint bandwidth at the DRAM interface and support all-to-all communication. The accelerator adopts a tiled-based design, (Fig. 2 shows a 4 × 4 tiles are connected through the flexible interconnect as an illustration).

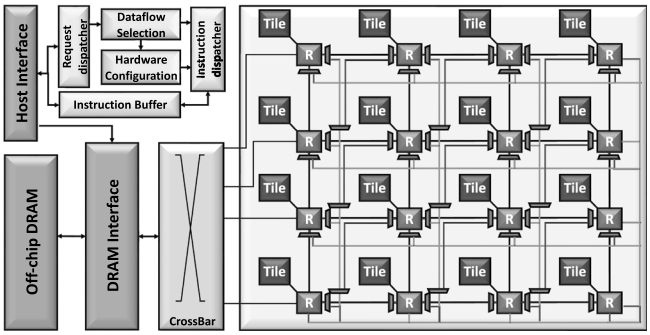

Fig. 2. Versa-DNN accelerator architecture (an example of 4 × 4 architecture).

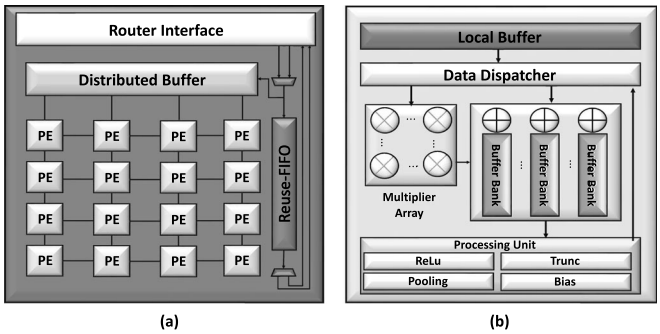

Fig. 3. (a) Proposed tile architecture, (b) proposed PE architecture.

B. Proposed Tile and PE Architecture

Each tile consists of a distributed buffer, a router interface, a spatial architecture with a 4 × 4 processing element (PE) array, and a reuse First-in-First-Out (FIFO) buffer, shown in Fig. 3(a). The reuse FIFO can temporarily store the data it received and send locally stored data to its neighbors, acting as a double buffer [11], which can support inter-tile communication. This is designed to enable the data exchange between distributed buffers, reducing off-chip memory access. As depicted in Fig. 3(b), each PE consists of a local buffer, a data dispatcher, an MAC array, and a processing unit (e.g., ReLu).

C. Flexible NoC Design

Even though altering dataflow can optimize the off-chip memory access, the distributions of matrices could also lead to various communication patterns. Previous work [12] has addressed the communication challenges raised by the dynamic dataflows in the unified global buffer, the communication behaviors among distributed buffers remains unexplored.

In general, the flexible NoC design enables data propagation between adjacent tiles. This can reduce the complexity of

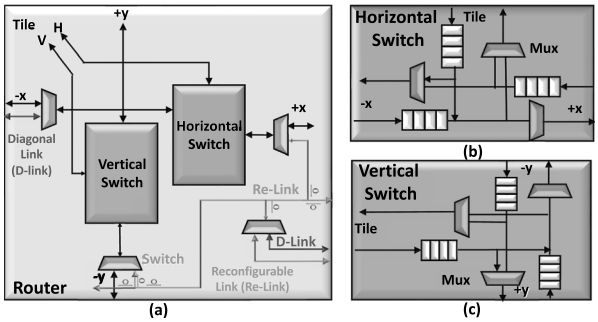

Fig. 4. (a) Proposed router architecture, (b) horizontal switch, and (c) vertical switch.

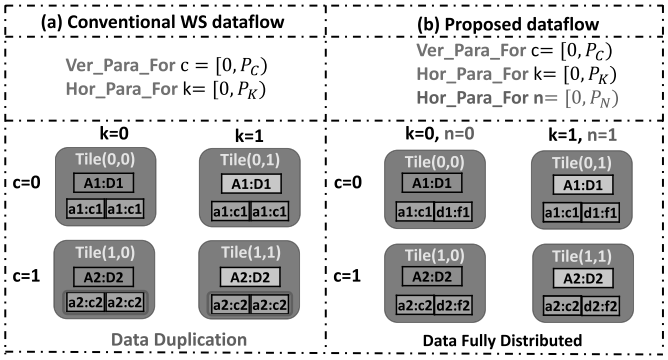

Fig. 5. (a) Nested loop representation of the parallelism strategy and mapping example of the conventional WS dataflow, (b) Nested loop representation of the parallelism strategy and mapping example of the proposed dataflow.

data exchange and avoid complicated communication protocols. Specifically, the proposed NoC design can be partitioned into multiple ring topologies with any size and location. These rings will be formed to propagate the weights and input activations. The proposed flexible NoC consists of a flexible router, reconfigurable links, and diagonal links, which will be further discussed in Fig. 4.

1) Flexible Router: The major functionality required for ring topology is to forward and eject/inject in-flight packets. This significantly simplifies the router design with much-reduced radix. As shown in Fig. 4(a), the flexible router consists of a vertical switch and a horizontal switch. The vertical switch processes column-wise communication, and the horizontal switch processes row-wise communication. Vertical switches are connected to reconfigurable links (called Re-link in Fig. 4), and horizontal switches are connected to Re-link and diagonal links (called D-link in Fig. 4). The Re-link consists of multiple simple transistors to turn on/off the link connection between two adjacent 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 residing across the diagonal, which can effectively reduce the communication distance and hop count and bridge non-adjacent routers.

IV. Proposed Flexible Dataflow

A variety of dataflows have been explored to reduce the off-chip memory access in current DNN accelerators, but very few of them can be effectively applied to an accelerator design with distributed buffers. The major issue with a distributed buffer setup is data duplication or complicated communication patterns. For example, Simba [13] proposed a weight-stationary dataflow in a chiplet-based DNN accelerator, and its dataflow could result in reduced inter-chiplet communications (given the limited inter-chiplet bandwidth) but at a considerable cost of data duplication at each chiplet. To address this limitation, designing a flexible dataflow with simplified communication and eliminating data duplication would require two elements:

- A thorough study and analysis of parallelism strategies that can minimize data duplication and reduce DRAM access.
- A dataflow selection algorithm that can select the optimal dataflow with the least DRAM access and data duplication.

A. Parallelism Strategies

To analyze the parallelism strategies or to answer the raised question, we need to examine the root cause of data duplication in existing designs.

Fig. 5(a) shows the nested loop representation of a parallelism strategy and mapping example of the tiled convolutional layer with weight-stationary (WS) dataflow [13], [14]. The tiled weight and input matrices are distributed into each distributed buffer. The number of partitioned matrices is 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 tile-based accelerator in the horizontal and vertical direction. Since attributes k and c are the indexes of 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 ($I_{[n][c][x][y]}$) is independent of k, and therefore, the input partition remains duplicated at each row. Because of the data duplication, the on-chip buffer cannot be effectively utilized. This further diminishes the on-chip data reuse opportunities, and thus it could affect the DRAM access.

To meet the mentioned dataflow 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 need to be distributed across each row and column.

We use a parallelism strategy depicted in Fig. 5(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 to store all the input, weight, and output matrices. If attributes c and n are parallelized in different directions (Horizontally and Vertically), the input matrix will be fully distributed across each row and column. Similarly, the weight matrix will be fully distributed, as c and k are the two weight matrix attributes. As a result, on-chip buffers can be fully utilized, and the proposed dataflow can effectively reduce off-chip memory access thanks to optimized data reuse. Our current analysis can be extended to any parallelism dimensions.

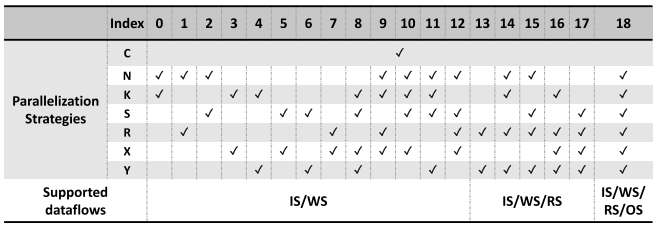

Fig. 6. All the supported dataflows in Versa-DNN accelerator.

If attributes C, K, R, and S are parallelized, the weight matrix will be fully distributed across each row and column, as these are four attributes of the weight matrix. Similarly, if attributes C, N, X, and Y are parallelized, the input matrix will be distributed across each row and column.

Several parallelism strategies are viable to mitigate the data duplication issue, but some of them come with complicated communication patterns. To further address the communication complexity problem, we aim to select those dataflows with simple data movement patterns like a ring. This requires that both input and weight matrices be partitioned with an equal number of indexes, and that attribute c be parallelized, as each channel is independent of others and there is no communication between input and weight matrices across different channels.

As a conclusion, the dataflow must meet the following three requirements: (1) at least two indexes of both weight and input matrices must be parallelized across each row and column, (2) both weight and input matrices must be partitioned with an equal number of indexes, and (3) attribute c must be parallelized. All the supported dataflow candidates are shown in Fig. 6.

B. Dataflow Selection

The primary goal of dataflow selection is to reduce off-chip memory access. The off-chip memory access is determined by two factors: the data volume of each invocation and the number of invocations. To better illustrate the problem, we use an example depicted in Fig. 5.

Recall that the DNN layer (l) 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 of attributes in each partitioned matrix is represented as i_{par}. For example, input channel C is partitioned into P_C parts, and each part has C_{par} elements.

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 types can be calculated as V_{wt}, V_{ifmap}, V_{psum} as depicted in (1)

$$V_{wt} = \prod i_{par} \times P_i, i \in \{K, C, S, R\}$$

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 $i^l, i \in \{K, C, R, S, N, X, Y\}, l \in [0, L)$
Input : Dimensions of the a sub-layer in distributed buffer $i^l_{par}, i \in \{K, C, R, S, N, X, Y\}, l \in [0, L)$
Input : Distributed buffer capacity C_{DB}
Output: $Parallelfactor : P^l_i, i \in \{K, C, R, S, N, X, Y\}, l \in [0, L)$
Output: $OptimalDataflow$

```
1  for l ∈ [0, L) do
2      for P_i^l ∈ DF do
3          // Data volume involved in each
               invocation
4          Calculate V_d^l, d ∈ {wt, ifmap, psum}, l ∈ [0, L);
5          // Number of invocations
6          Calculate R_d^l, d ∈ {wt, ifmap, psum}, m ∈ [0, L);
7          // Number of invocations
8          DA^l = Σ_d V_d × R_d, d ∈ {wt, ifmap, psum}, l ∈
             [0, L);
9          for datavolume^l ∈ [0, C_DB] do
10             Find the Minimal DA^l;
11         end
12         return P_i^l;
13     end
14     return Optimal Dataflow
15 end
```

$$V_{ifmap} = \prod i_{par} \times P_i, i \in \{N, C, X, Y\}$$
$$V_{psum} = \prod i_{par} \times P_i, i \in \{N, K, X', Y'\}. \quad (1)$$

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 the concept can be generalized to support output/input/row stationary dataflow. The total number of invocations (R_d) of weight stationary dataflow can be calculated as depicted in (2)

$$R_{wt} = \prod \frac{i}{i_{par} \times P_i}, i \in \{K, C, S, R\}$$
$$R_{ifmap} = \prod \frac{i}{i_{par} \times P_i}, i \in \{N, K, C, S, R, X', Y'\}$$
$$R_{psum} = \frac{\prod^{i \in \{N,K,S,R,X',Y'\}} i \times (\frac{2C}{C_1 \times P_C} - 1)}{\prod^{i \in \{N,K,S,R,X',Y'\}} i_{par} \times P_i}. \quad (2)$$

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

$$\begin{aligned} \text{datavolume} = & \prod i_{par}, i \in \{K, C, S, R\} \\ & + \prod i_{par}, i \in \{N, C, X, Y\} \\ & + \prod i_{par}, i \in \{N, C, X', Y'\}. \end{aligned} \quad (3)$$

The data volume of the partitioned matrices is the accumulation of data volume for weight matrix, input activation matrix, and

output activation matrix. The data volume of weight matrix are the product of data volume for multiple attributes including K, C, S, and R in each partitioned matrix. The data volume of input activation matrix are the product of data volume for multiple attributes including N, C, X, and Y in each partitioned matrix. The data volume of output activation matrix are the product of data volume for multiple attributes including N, C, X', and Y' in each partitioned matrix. 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 access with the distributed buffer capacity(C_{DB}) limitation.

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

V. Dynamic Accelerator Scheduler

Although pipelining DNN layers [15] could reduce data duplication, the inter-layer dependency could still incur resource underutilization. Each DNN layer has to wait until the completion of the prior layer. In light of this, simultaneously running multiple independent DNN applications could provide better parallelism. This opens up a new question - how shared computing units, buffers, and NoCs should be effectively allocated among multiple DNN applications?

We propose a scheduling policy to allocate computation and communication resources for running DNN tasks (i.e., independent DNN layers). The scheduler will allocate the accelerator resources to running DNN tasks when any task is completely executed or a new task is added. We note that, whenever a new DNN is dispatched from the host to the accelerator, the DNN model is assigned a fixed user-defined priority. The priority is statically predetermined as a configuration parameter [16]. The key idea of the partition algorithm is to fully utilize the accelerator's computation resources by (1) creating independent DNN tasks from multiple DNN applications, (2) scheduling the candidate tasks during runtime to leverage the flexibility of interconnects and co-location of multiple DNN models, and (3) executing them in parallel.

The scheduler follows a two-step procedure: (1) the Candidate Task Selection that decides which candidate layers to execute next (Algorithm 2 Line 2–11), and (2) once the candidate tasks are chosen, the scheduler determines the allocation of resource (number of tiles) by Allocation Policy and considers both the task's running time and its priority level to balance latency and throughput satisfaction (Algorithm 2 Line 13–31).

First, among all multiple DNN models (represented by m) that have been dispatched to the scheduler, the scheduler divides each of them into multiple tasks (represented by l) and selects a group of tasks as candidates to execute according to the directed acyclic graph (DAG). A task can be selected as part of the candidate group if it is independent of all other tasks in the candidate group. The candidate group is updated when any task completes its execution or a new task is added.

Algorithm 2: Versa-DNN Scheduler.

Input : Workload features:
Layers in each DNN model ($m_l, m \in [0, M), l \in [0, L)$),
Number of MACs per layer ($MACs^l_m$),
Tile utilization per layer (U^l_m),
DRAM access per layer ($DRAM^l_m$),
Sizes of reused data (iS^l_m, wS^l_m, pS^l_m,),
Priority of each DNN model ($Pri_m, m \in [0, M)$).

Input : Hardware features:

Total number of available tiles ($TotalTiles$),
Number of MAC unit in each tile ($MUnit$),
Operation frequency rate (F),
Available on-chip bandwidth for different data types (iB^l_m, wB^l_m, pB^l_m,),
DRAM memroy bandwidth ($OffBw$),
Priority of each DNN model ($Pri_m, m \in [0, M)$).

Output: $Allocation[]$

1 /* Generate the candidate group needed to process */
2 **Function** *Candidate tasks Selection*
3 // Divide the dispatched models into multiple tasks.
4 **for** $m \in [0, M)$ **do**
5 $m_l \leftarrow m, l \in [0, L)$;
6 **end**
7 // Choose dependency free tasks according to the DAGs of multiple DNN models
8 **for** $m \in [0, M), l \in [0, L)$ **do**
9 $Candidate[] \leftarrow DependencyFree(m_l)$;
10 **end**
11 **return** $Candidate[]$;
12 /* Allocate hardware resource */
13 **Function** *Allocation Policy*
14 $IsolatedTime \leftarrow []$;
15 $Score.Candidate \leftarrow []$;
16 $Allocation \leftarrow []$;
17 **for** $m \in [0, M)$ **do**
18 $Score[m_l] \leftarrow Pri_m$;
19 **end**
20 **for** $task \in Candidate[]$ **do**
21 $IsolatedTime[task] \leftarrow TimePredict[task]$;
22 $TimePredict[task] \leftarrow \max(TimeOn[task], TimeOff[task])$;
23 $TimeOn[task] \leftarrow \max(OnComp[task], OnComm[task])$;
24 $OnComp[task] \leftarrow \frac{MACs[task]}{U[task] \times F \times MUnit \times Totaltiles}$;
25 $OnComm[task] \leftarrow \max(\frac{iS[task]}{iB[task]}, \frac{wS[task]}{wB[task]}, \frac{pS[task]}{pB[task]})$;
26 $TimeOff[task] \leftarrow \frac{DRAM[task]}{OffBw}$;
27 $Score.Candidate[task] \leftarrow \frac{Score[task]}{IsolatedTime[task]}$;
28 $fraction \leftarrow \frac{Score.Candidate[task]}{\sum Score.Candidate}$;
29 $Allocation[task] \leftarrow \lfloor fraction \times TotalTiles \rfloor$;
30 **end**
31 **return** $Allocation[]$;

Second, the scheduler allocates the resource by considering both the task running time and model priority level to balance latency, throughput, and fairness satisfaction. The scheduler grants an initial number of scores per its priority level for each DNN model, which is statically predetermined as a configuration parameter. The scheduler assigns an initial score to tasks based on the priority level of the DNN model they belong to, as indicated in Algorithm 2 (Line 18). Then Algorithm 2 adjusts the number of scores to offer scheduling opportunities to the layers with low priority but have short execution time.

The scheduler distributes these spare resources to running tasks by following a score function depicted in Algorithm 2 (Line 27), and it balances the execution time, communication, and priority satisfaction of each candidate layer.

The score function is derived by comparing the initial score ($Score[task]$) against its isolated execution time ($IsolatedTime[task]$), and the scheduler grants $Isolated\ Time[task]$ per task's execution time ($TimePredict[task]$) with all valid computation resources and without interruption shown in Algorithm 2. The isolated execution time is predetermined by the compiler, as a roofline model [7] can estimate the $TimePredict[task]$ that can be expressed as a function of on-chip execution time ($TimeOn[task]$) and Off-chip DRAM access time ($TimeOff[task]$) as follows:

$$TimePredict[task] \leftarrow \max(TimeOn[task], TimeOff[task]). \quad (4)$$

The on-chip execution time can be expressed as a function of on-chip computation time ($OnComp[task]$) and on-chip communication time ($OnComm[task]$)

$$TimeOn[task] \leftarrow \max(OnComp[task], OnComm[task]). \quad (5)$$

On-chip computation time: We model $OnComp[task]$ as follows:

$$OnComp[task] \leftarrow \frac{MACs[task]}{U[task] \times F \times MUnit \times Totaltiles}. \quad (6)$$

Where U is the tile utilization rate, F is the operating frequency of the accelerator, $MUnit$ is the number of MAC unit in each tile, and $Totaltile$ means the total number of tile.

On-chip communication time: We model $OnComm[task]$ as follows:

$$OnComm[task] \leftarrow \max\left(\frac{iS[task]}{iB[task]}, \frac{wS[task]}{wB[task]}, \frac{pS[task]}{pB[task]}\right). \quad (7)$$

Where $iS[task], wS[task], pS[task]$ are the sizes of different reused data, $iB[task], wB[task], pB[task]$ are the available on-chip bandwidths for different data types.

Off-chip DRAM access time: Total time for each layer ($TimeOff[task]$), includes the time needed to read input data (input feature and filter weights) from DRAM to scratchpad, and the time needed to write the output feature back from the scratchpad to DRAM. We then express the $TimeOff[task]$ as follows:

$$TimeOff[task] \leftarrow \frac{DRAM[task]}{OffBw}, \quad (8)$$

where $DRAM[task]$ is the total DRAM access of each task obtained by Algorithm 1 and $OffBw$ means the total DRAM memory bandwidth.

This allows even low-priority tasks to gradually increase their scores and get a chance to allocate more computation resources. Consequently, this score function not only fosters throughput but also the priority among the candidate tasks. Finally, the scheduler allocates the spare resources ($TotalTiles$) proportional to the score of each task, where the allocated number of tiles is rounded down (not up) to the closest integer, shown in Algorithm 2.

VI. Dynamic Sub-Accelerator Partition and Hardware Reconfiguration

In this section, we describe the dynamic partitioning and hardware configuration in support of flexible dataflow for multi-DNN execution. As mentioned, the task scheduler and dataflow selection determine the overall resource allocation and parallelism strategies. Upon the decision, versa-DNN will be dynamically partitioned into several sub-accelerators, each optimized for its optimal dataflow.

Essentially, the sub-accelerator partitioning is to partition the computation, memory, and communication modules. As the computation and memory modules are distributed, the difficulty is to partition the NoC and configure it to support various dataflows. As such, the NoC of the sub-accelerator will be partitioned into multiple disjoint subNoCs. Each NoC will be configured to support the communication patterns of the selected dataflow method. 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 differences among these dataflows include the data movement patterns of input, weight, and partial sums. For example, in Fig. 7(a), 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. For RS dataflow shown in Fig. 7(b), inputs are communicated horizontally via those rings, and partial sums are accumulated via diagonal links. For output-stationary dataflow shown in Fig. 7(b), the weight matrices are propagated horizontally, whereas the input matrices move vertically.

For these examples, 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 same attributes C and Y will be clustered into 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 same ring to transfer input activations and each partition with the same attributes S and R will be clustered into same ring to forward weights. Within each ring, each tile will forward the data to its bottom tile given a simpler routing algorithm. This naturally avoids the deadlock while maintaining the simplicity of the routing.

Network deadlock can occur due to protocol dependency or circular channel dependency. Specifically, the protocol dependency is eliminated by separating the request and reply packets to different virtual networks so that the protocol deadlock is avoided. To prevent circular channel dependency [17], [18], [19], we use the simple yet effective dateline [17].

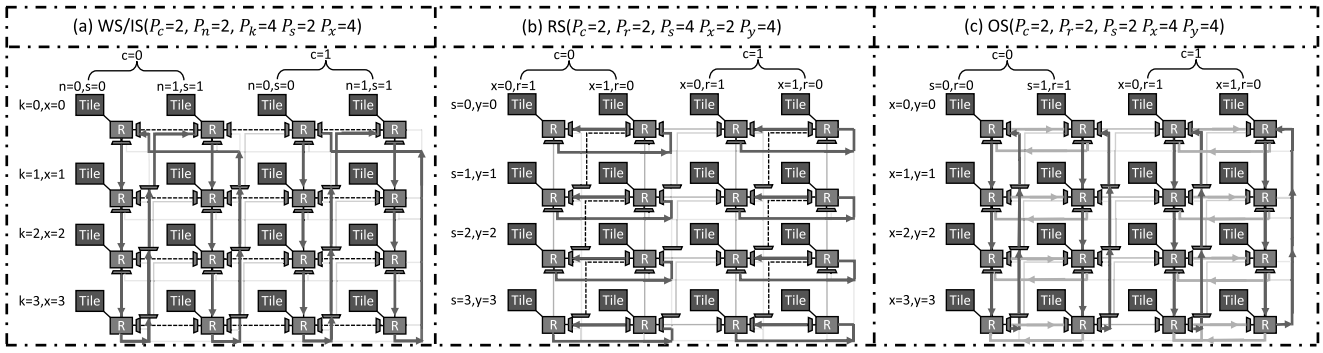

Fig. 7. Topology configurations of (a) weight stationary/input stationary dataflow, (b) row stationary dataflow, and (c) output stationary dataflow with different parallelism strategies.

TABLE I
MULT-DNN WORKLOADS

Workloads		DNN Model
Workload A Similar to Planaria	Workload A-1	ResNet-50 [22], GoogLeNet [23], YOLOv3 [24], SSD-R [25], and GNMT [26]
	Workload A-2	EfficientNet-B0 [27], MobileNet-v1 [28], SSD-M [25], and Tiny YOLO [29]
	Workload A-3	ResNet-50 [22], GoogLeNet [23], EfficientNet-B0 [27], MobileNet-v1 [28], YOLOv3 [24], SSD-ResNet34 [25], SSD-MobileNet [25], Tiny YOLO [29], and GNMT [26]
Workload B Similar to Herald	Workload B-1	ResNet-50 [22], Unet [30], and MobileNet-v2 [31]
	Workload B-2	ResNet-50 [22], Unet [30], MobileNet-v2 [31], BR-Q Handpose [32], and Focal Length DepthNet [33]
	Workload B-3	ResNet-50 [22], MobileNet-v1 [28], SSD-ResNet34 [25], SSD-MobileNet-v1 [25], and GNMT [26]
Workload C Similar to AI-MT	Workload C-1	VGG-16 [34], ResNet-50 [22], and MobileNet-v1 [28]
	Workload C-2	GNMT [26], ResNet-50 [22], ResNet-34 [22], and MobileNet-v1 [28]
	Workload C-3	VGG-16 [34], GNMT [26], ResNet-50 [22], ResNet-34 [22], and MobileNet-v1 [28]

VII. EVALUATION

A. Simulation Setup

We extend Timeloop [20] simulator to support the non-uniform latency and bandwidth between tiles, and the non-uniform latency and bandwidth between PEs. The extended simulation framework is capable of capturing various accelerator activities, such as arithmetic operation, memory access at different levels, and NoC latency and bandwidth. The simulator also considers the consequences of the application of various dataflows and system configurations. The energy consumption of these DRAM read/write operations is modeled by [21].

Multi-DNN workloads: For a fair comparison, we reproduced similar benchmark applications as the ones used in Planaria [6], Herald [1], and AI-MT [5], shown in Table I.

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, 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 700 MHz. The on-chip distributed buffer capacity of each tile is 100 KB. For a fair comparison, we keep the configurations to be consistent for all compared designs (Planaria [6], Herald [1], and AI-MT [5]): All designs use 16384 processing elements, and each processing element contains 16 MAC units, the SRAM of each processing element is 5 KB and the total on-chip SRAM capacity is 100 MB.

Dataflow Modeling: For our evaluations, we use our proposed dataflow for the proposed accelerator Versa-DNN. The proposed dataflow is fully distributed and has an optimal parallel policy at the tile level, determined by Algorithm 1 according to different workloads requirements and hardware resource allocation. We use conventional weight stationary dataflow(WS) [14], row stationary dataflow(RS) [4], and output stationary dataflow(OS) [35] for comparison. We implement all compared dataflows on a systolic array-based accelerator platform and our proposed dataflow on the proposed Versa-DNN accelerator to act as a good test for our proposed dataflow to learn and exploit their difference.

Interconnect Configuration: We use the proposed hardware reconfiguration mentioned in Section VI for the configuration of the proposed accelerator. The Interconnection of AI-MT [5] and Herald [1] are fixed. The AI-MT [5] consists of several systolic arrays of identical and Herald [1] consists of several heterogeneous sub-accelerators that are optimized for conventional dataflows (WS, OS, and RS). We use the Manual-tuned method for the dynamic architecture fission of Planaria [6].

B. On-Chip Distributed Buffer Utilization Analysis

Fig. 8 illustrates the time stamp of normalized efficient on-chip distributed buffer capacity utilization of conventional WS, RS, OS, the average of compared conventional dataflows, and proposed dataflow in a single layer of DNN model VGG-16. The proposed dataflow has higher utilization of on-chip distributed buffers in each time stamp compared to conventional WS, OS, and RS. Fig. 9 illustrates the normalized efficient on-chip distributed buffer capacity utilization for conventional

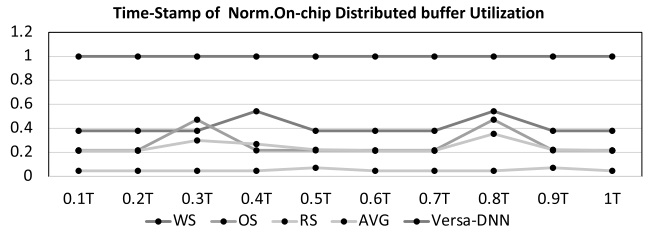

Fig. 8. Time stamp of normalized efficient on-chip distributed buffer capacity utilization of conventional weight stationary dataflow(WS) [14], row stationary dataflow(RS) [4], output stationary dataflow(OS) [35], the average of compared conventional dataflows, and proposed dataflow in a single layer of DNN model VGG-16 [34]. T is the execution time of a single layer of DNN model VGG-16.

WS [14], RS [4], OS [35], the average of compared conventional dataflows, and the proposed dataflow for each convolutional layer of the DNN model VGG-16 [34]. To improve the efficient utilization of on-chip distributed buffers, the proposed dataflow eliminates the data duplication that exists in conventional dataflows. The proposed dataflow increases the efficient utilization of on-chip distributed buffers for each convolutional layer of the DNN model VGG-16 by an average of 68% compared to WS, RS, and OS. The proposed dataflow can hold more data on-chip, which can help reduce excessive DRAM accesses (discussed in the next subsection).

C. DRAM Analysis

Fig. 10 illustrates the normalized DRAM Access for conventional WS [14], RS [4], OS [35], the average of compared conventional dataflows, and the proposed dataflow for each convolutional layer of the DNN model VGG-16 [34]. The DRAM accesses include the off-chip DRAM accesses for weights, input activations, and output activations.

The conventional weight stationary dataflow [14] parallelizes over channel and kernel dimensions, and thus it is highly efficient for late layers in CNN-based models. The conventional row stationary dataflow [4] parallelizes across weight and input activation dimensions and excels on the early layers of CNN-based models. The conventional output stationary dataflow parallelizes on output activation dimensions.

The proposed dataflow outperforms the conventional dataflows in a single DNN model because of its higher on-chip distributed buffer utilization (discussed in the last subsection). Furthermore, conventional dataflows cannot adapt to various parallelism policies and tiling factors. More importantly, they are designed for centralized buffers and have limited applicability to distributed buffer settings. As such, in a single DNN model, the proposed dataflow reduces DRAM Access by an average of 71.9% in each convolutional layer of VGG-16 [34], compared to the average of conventional weight stationary dataflow, row stationary dataflow, and output stationary dataflow.

For the same reason, our proposed design reduces DRAM access in multi-DNN workload scenarios. Fig. 11 shows the normalized DRAM access for each workload scenario using our proposed accelerator compared to other accelerators (Planaria [6], Herald [1], and AI-MT [5]). In multi-DNN workload scenarios A1, A2, and A3, our proposed acceleration platform reduces DRAM access by 35%, 42%, and 17%, respectively, compared to the Planaria accelerator [6]. In scenarios B1, B2, and B3, our proposed accelerator, Versa, reduces DRAM access by 40%, 42%, and 48%, respectively, compared to the Herald accelerator [1]. In scenarios C1, C2, and C3, the proposed accelerator reduces DRAM access by 38%, 21%, and 32%, respectively, compared to the AI-MT accelerator [5].

D. Throughput Analysis

Throughput is the main performance metric for the evaluation of server scenarios for inference tasks in MLPerf [36] and AR/VR [37]. Fig. 12 compares the throughput of the proposed design with Planaria [6], Herald [1], and AI-MT [5] across various workload scenarios while meeting the priority requirements. The proposed design outperforms the compared accelerator platforms in all workload scenarios. Herald [1] executes the DNN layers sequentially, but AI-MT [5], Planaria [6], and the proposed design run multiple independent sublayers or layers in parallel, which can fully utilize the on-chip hardware resource and layer-wise parallelism. All baseline designs utilize fixed parallelism and were originally designed for centralized buffers. As a result, these designs may lead to data duplication when used with distributed buffers.

Our proposed architecture can support multiple fully distributed dataflows with an optimal parallelism strategy that can eliminate data duplication and minimize DRAM access at runtime. The proposed acceleration platform increases throughput by 1.13×, 1.32×, and 1.80× compared to the compared acceleration platform Planaria [6] in multi-DNN workload scenarios A1, A2, and A3, respectively. The performance gains of our proposed design come from better hardware utilization provided by the adaptive dataflow design, as each layer of those DNN models can be optimized individually. In contrast, the hardware resources of prior work are underutilized due to inadequate dataflows in workloads A1, A2, and A3.

The proposed acceleration platform increases throughput by 3.33×, 2.04×, and 4.76× compared to the compared acceleration platform Herald [1] in multi-DNN workload scenarios B1, B2, and B3, respectively. This trend can be traced back to the capabilities of the proposed accelerator compared to Herald [1]. Unlike Herald, the proposed accelerator is capable of executing different layers from different models concurrently through the implementation of a scheduling Algorithm 2. The DNN models in workload scenarios B1, B2, and B3 involve significant amounts of depth-wise/point-wise convolutions, where data-level parallelism has very limited performance. The proposed design demonstrates superior performance in resource utilization for these types of convolutions by simultaneously running multiple independent tasks to exploit the task-level parallelism.

The proposed acceleration platform increases throughput by 6.06×, 4.85×, and 3.85× when compared to AI-MT [5], in multi-DNN workload scenarios C1, C2, and C3, respectively. Even though AI-MI can support the simultaneous execution of multiple DNN layers, it still suffers from resource underutilization caused by the rigid PE design and fixed dataflow. This issue

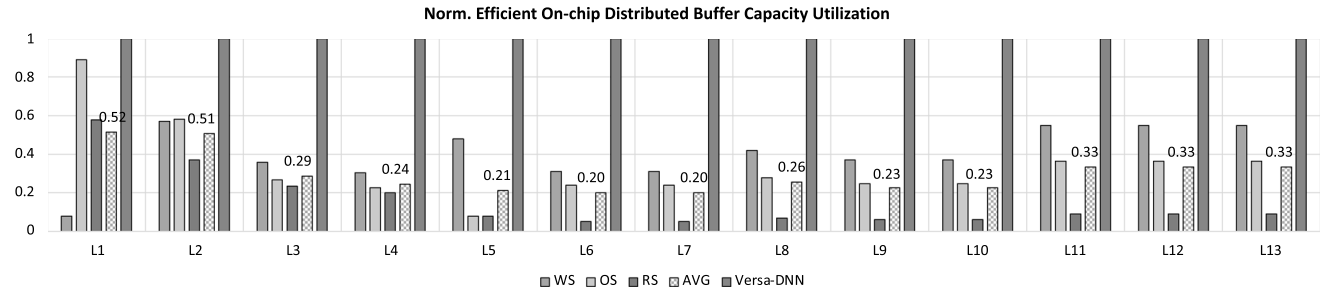

Fig. 9. Normalized efficient on-chip distributed buffer capacity utilization of conventional weight stationary dataflow(WS) [14], row stationary dataflow(RS) [4], output stationary dataflow(OS) [35], the average of compared conventional dataflows, and proposed dataflow in each convolutional layer of DNN model VGG-16 [34], normalized to efficient on-chip distributed buffer capacity utilization of proposed dataflow.

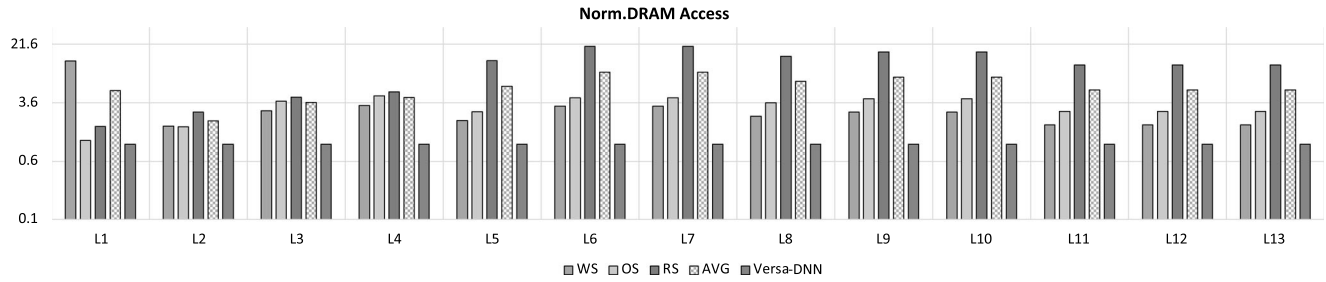

Fig. 10. Normalized DRAM Access of conventional weight stationary dataflow(WS) [14], row stationary dataflow(RS) [4], output stationary dataflow(OS) [35], the average of compared conventional dataflows, and proposed dataflow in each convolutional layer of DNN model VGG-16 [34], normalized to DRAM Access of proposed dataflow.

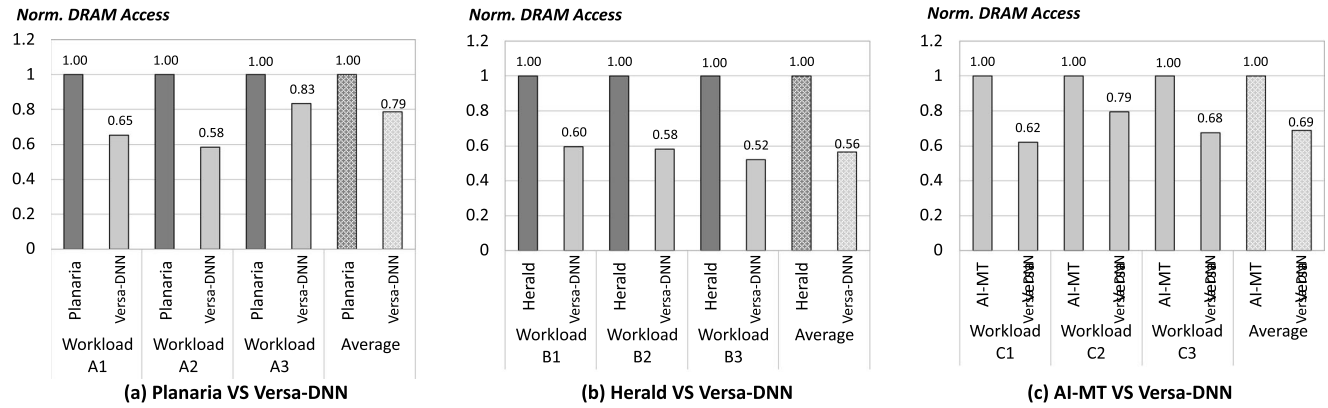

Fig. 11. Normalized DRAM Access for each workload scenario by using our proposed accelerator and compared accelerators ((a) Planaria [6], (b) Herald [1], and (c) AI-MT [5]), normalized to DRAM Access of compared accelerators respectively.

is addressed in our proposed design via the architecture and dataflow flexibility.

E. Energy Consumption Analysis

For energy analysis, we utilize a customized version of the open-source Timeloop simulator [20] to obtain power consumption and execution time. It should be noted that the evaluation takes into account the energy consumption of the entire system, including control units, computation units, DRAM, distributed buffer, local buffer, and interconnects. Fig. 13 presents the normalized overall energy consumption analysis of the proposed accelerator. As can be seen, the proposed acceleration platform increases throughput by 24%, 33%, and 32% compared to the Planaria platform [6] in multi-DNN workload scenarios A1, A2, and A3, respectively. The proposed acceleration platform increases throughput by 58%, 47%, and 67% compared to the Herald platform [1] in multi-DNN workload scenarios B1, B2, and B3, respectively. The proposed acceleration platform increases throughput by 68%, 60%, and 58% compared to the AI-MT platform [5] in multi-DNN workload scenarios C1, C2, and C3, respectively. The main factors contributing to these reductions are the reduction in DRAM access (due to the proposed dataflow) and the dataflow selection Algorithm 1, a simple interconnect,

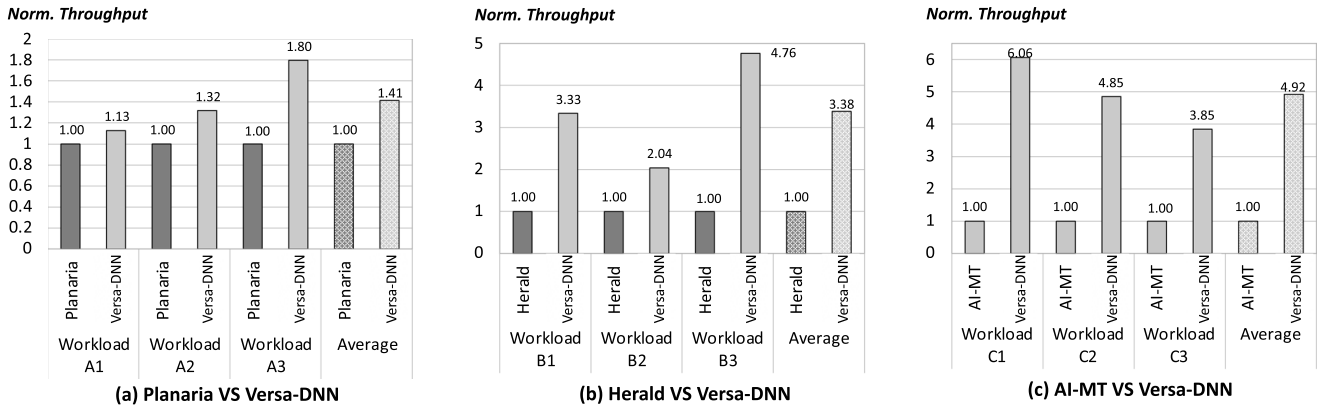

Fig. 12. Normalized throughput for each workload scenario by using our proposed accelerator and compared accelerators ((a) Planaria [6], (b) Herald [1], and (c) AI-MT [5]), normalized to throughput of compared accelerators respectively.

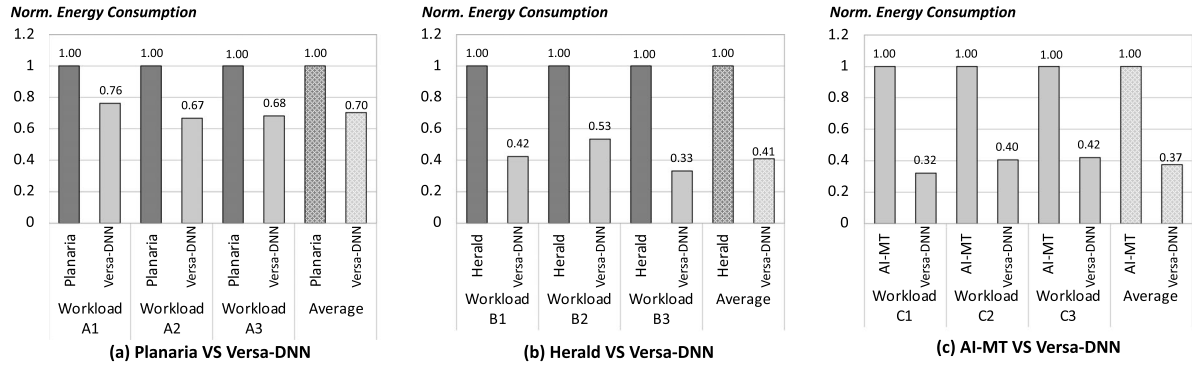

Fig. 13. Normalized energy consumption for each workload scenario by using our proposed accelerator and compared accelerators ((a) Planaria [6], (b) Herald [1], and (c) AI-MT [5]), normalized to the energy consumption of compared accelerators respectively.

reduced long-distance communication (owing to the multiple dimensions parallelism strategy), and low configuration time (attributed to hardware reconfiguration).

F. Area Analysis

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

VIII. Related Work

A. Heterogeneous NoC Designs

In addition to reconfigurable NoC topologies, prior research [38], [39], [40], [41], [42], [43] has been proposed to combine the benefits of various topologies in a holistic manner. A hierarchical ring topology, with local and global rings, is designed to take both local and global communication into consideration. Asit et al. [42] designed a heterogeneous NoC consisting of two subnetworks with optimized bandwidth and latency. In addition to these heterogeneous designs, high-radix on-chip networks [39], [41] often deploy both concentration and bypassing techniques to keep the NoC latency scaling to increased NoC size while maintaining the area and wiring costs. However, these designs only provide the flexibility in space but have restricted flexibility in handling diverse communication behaviors at runtime.

B. Optimized DNN Dataflow

In the field of hardware acceleration for machine learning, various dataflows have been proposed to optimize data access and communication patterns. Chen et al. performed a classification of neural network accelerators based on their data reuse characteristics, including weight stationary, output-stationary, and row-stationary dataflows. These dataflows determine the mapping of data to Processing Elements (PEs) in different ways, which affects the spatial and temporal utilization of data. Furthermore, Yang et al. [12], [44], [45], [46] broadened the scope of the dataflows utilized in contemporary Neural Network

(NN) accelerators by providing a comprehensive taxonomy of these dataflows. This work underscores the importance of understanding the interplay between dataflow design and hardware acceleration for NN computation.

C. Multi-DNN Accelerators

Several DNN accelerators with varying dataflows have been proposed in recent years. These accelerators are implemented on a monolithic processing array and focus on improving data reuse, assuming uniform latency and bandwidth across all processing elements (PEs). However, the increasing size of DNN models has led to the development of scale-up systems, such as distributed architectures. Shao et al. implemented a DNN on a distributed architecture using an electrical mesh network for inter-tile communication. Despite using aggressive electrical wire technology, this approach still becomes a performance bottleneck as the system scales, highlighting the scalability limitations of electrical networks at the chiplet scale. Shen et al. investigated the use of multiple FDA sub-accelerators, which they referred to as convolutional layer processors, running the same dataflow in FPGAs [47]. PREMA [7] developed a scheduling algorithm for preemptive execution of DNNs on a monolithic accelerator and utilized time-sharing for multi-DNN. AI-MT [5], on the other hand, developed an architecture that supports multi-DNN by first tiling the layers at compile time and then exploiting hardware-based scheduling to maximize resource utilization. This approach employed multiple systolic arrays within an accelerator chip and parallelized computation tiles of each layer. Planaria [6] explored the design of an architecture for spatial co-location of DNNs for multi-DNN acceleration and addressed its unique scheduling challenges.

IX. Conclusion

In this paper, we propose Versa-DNN, a versatile DNN accelerator that can provide efficient computation, memory, and communication support for the simultaneous execution of multiple DNNs, which is also suitable for NLP tasks. The Versa-DNN features three unique designs: a flexible off-chip memory access optimization strategy, adaptable communication fabrics and distributed buffers, and a communication-aware scheduling algorithm. The proposed off-chip memory access optimization strategy can improve the performance and energy efficiency by increasing hardware utilization, eliminating excess data duplication (we never mention this before), and reducing off-chip memory accesses. The adaptable fabrics consist of distributed buffers, processing elements, and a flexible Network-on-Chip (NoC), which can dynamically morph and fission to support distinct communication and computation needs for various simultaneously running DNN models. Furthermore, the proposed communication-aware scheduling policy orchestrates the simultaneous execution of multiple DNN models with improved performance and energy efficiency. The Versa-DNN is evaluated using same benchmark DNN models as compared state-of-art accelerators, and the simulation results demonstrate that our proposed Versa-DNN architecture achieves 41%, 238%, 392% throughput speedup and 30%, 59%, 63% energy reduction on average for different workloads when compared to Planaria, Herald, and AI-MT, respectively.

Acknowledgments

The authors would like to thank the anonymous reviewers for their insightful comments.

References

[1] H. Kwon, L. Lai, M. Pellauer, T. Krishna, Y.-H. Chen, and V. Chandra, "Heterogeneous dataflow accelerators for multi-DNN workloads," in *Proc. IEEE Int. Symp. High Perform. Comput. Architecture*, 2021, pp. 71–83.

[2] H. Kwon, P. Chatarasi, M. Pellauer, A. Parashar, V. Sarkar, and T. Krishna, "Understanding reuse, performance, and hardware cost of DNN dataflow: A data-centric approach," in *Proc. IEEE/ACM Int. Symp. Microarchitecture*, 2019, pp. 754–768.

[3] S. I. Venieris, C.-S. Bouganis, and N. D. Lane, "Multi-DNN accelerators for next-generation ai systems," 2022, *arXiv:2205.09376*.

[4] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE J. Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, Jan. 2017.

[5] E. Baek, D. Kwon, and J. Kim, "A multi-neural network acceleration architecture," in *Proc. ACM/IEEE 47th Int. Symp. Comput. Archit.*, 2020, pp. 940–953.

[6] S. Ghodrati et al., "Planaria: Dynamic architecture fission for spatial multi-tenant acceleration of deep neural networks," in *Proc. IEEE/ACM 53rd Int. Symp. Microarchitecture*, 2020, pp. 681–697.

[7] Y. Choi and M. Rhu, "PREMA: A predictive multi-task scheduling algorithm for preemptible neural processing units," in *Proc. IEEE Int. Symp. High Perform. Comput. Architecture*, 2020, pp. 220–233.

[8] S.-C. Kao and T. Krishna, "MAGMA: An optimization framework for mapping multiple DNNs on multiple accelerator cores," in *Proc. IEEE Int. Symp. High Perform. Comput. Architecture*, 2022, pp. 814–830.

[9] Z. Liu, J. Leng, Z. Zhang, Q. Chen, C. Li, and M. Guo, "VELTAIR: Towards high-performance multi-tenant deep learning services via adaptive compilation and scheduling," in *Proc. ACM 27th Int. Conf. Archit. Support Program. Lang. Operating Syst.*, 2022, pp. 388–401.

[10] Y. H. Oh et al., "Layerweaver: Maximizing resource utilization of neural processing units via layer-wise scheduling," in *Proc. IEEE Int. Symp. High Perform. Comput. Architecture*, 2021, pp. 584–597.

[11] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-based accelerator design for deep convolutional neural networks," in *Proc. ACM/SIGDA Int. Symp. Field- Program. Gate Arrays*, 2015, pp. 161–170.

[12] J. Yang, H. Zheng, and A. Louri, "Adapt-flow: A flexible DNN accelerator architecture for heterogeneous dataflow implementation," in *Proc. ACM Great Lakes Symp. VLSI*, 2022, pp. 287–292.

[13] Y. S. Shao et al., "Simba: Scaling deep-learning inference with multi-chip-module-based architecture," in *Proc. IEEE/ACM 52nd Int. Symp. Microarchitecture*, 2019, pp. 14–27.

[14] NVDLA deep learning accelerator, 2017. [Online]. Available: http://nvdla.org

[15] M. Gao, X. Yang, J. Pu, M. Horowitz, and C. Kozyrakis, "TANGRAM: Optimized coarse-grained dataflow for scalable NN accelerators," in *Proc. ACM Int. Conf. Archit. Support Program. Lang. Operating Syst.*, 2019, pp. 807–820.

[16] P. Minet, E. Renault, I. Khoufi, and S. Boumerdassi, "Analyzing traces from a Google data center," in *Proc. IEEE Int. Wireless Commun. Mobile Comput. Conf.*, 2018, pp. 1167–1172.

[17] W. J. Dally and B. P. Towles, *Principles and Practices of Interconnection Networks*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003.

[18] C. Carrión, R. Beivide, J. Gregorio, and F. Vallejo, "A flow control mechanism to avoid message deadlock in k-ary n-cube networks," in *Proc. IEEE 4th Int. Conf. High-Perform. Comput.*, 1997, pp. 322–329.

[19] S. Ma, Z. Wang, Z. Liu, and N. E. Jerger, "Leaving one slot empty: Flit bubble flow control for torus cache-coherent NoCs," *IEEE Trans. Comput.*, vol. 64, no. 3, pp. 763–777, Mar. 2015.

[20] A. Parashar et al., "Timeloop: A systematic approach to DNN accelerator evaluation," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw.*, 2019, pp. 304–315.

[21] R. Gonzalez and M. Horowitz, "Energy dissipation in general purpose microprocessors," *IEEE J. Solid-State Circuits*, vol. 31, no. 9, pp. 1277–1284, Sep. 1996.

[22] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.
[23] C. Szegedy et al., "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 1–9.
[24] J. Redmon and A. Farhadi, "YOLOv3: An incremental improvement," 2018, *arXiv:1804.02767*.
[25] W. Liu et al., "SSD: Single shot multibox detector," in *Proc. 14th Eur. Conf. Comput. Vis.*, Amsterdam, The Netherlands, 2016, pp. 21–37.
[26] Y. Wu et al., "Google's neural machine translation system: Bridging the gap between human and machine translation," 2016, *arXiv:1609.08144*.
[27] M. Tan and Q. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 6105–6114.
[28] A. G. Howard et al., "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv:1704.04861*.
[29] J. Redmon and A. Farhadi, "YOLO9000: Better, faster, stronger," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 7263–7271.
[30] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Proc. Int. Conf. Med. Image Comput. Comput.-Assisted Intervention*, Munich, Germany, 2015, pp. 234–241.
[31] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 4510–4520.
[32] M. Madadi, S. Escalera, X. Baró, and J. Gonzalez, "End-to-end global to local CNN learning for hand pose recovery in depth data," 2017, *arXiv:1705.09606*.
[33] L. He, G. Wang, and Z. Hu, "Learning depth from single images with deep neural network embedding focal length," *IEEE Trans. Image Process.*, vol. 27, no. 9, pp. 4676–4689, Sep. 2018.
[34] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*.
[35] Z. Du et al., "ShiDianNao: Shifting vision processing closer to the sensor," in *Proc. ACM 42nd Int. Symp. Comput. Archit.*, 2015, pp. 92–104.
[36] V. Janapa Reddi et al., "MLPerf inference benchmark," 2019, *arXiv:1911.02549*.
[37] C.-J. Wu et al., "Machine learning at Facebook: Understanding inference at the edge," in *Proc. IEEE Int. Symp. High Perform. Comput. Architecture*, 2019, pp. 331–344.
[38] R. Das, S. Eachempati, A. K. Mishra, V. Narayanan, and C. R. Das, "Design and evaluation of a hierarchical on-chip interconnect for next-generation CMPs," in *Proc. IEEE Int. Symp. High Perform. Comput. Architecture*, 2009, pp. 175–186.
[39] J. Kim, J. Balfour, and W. Dally, "Flattened butterfly topology for on-chip networks," in *Proc. IEEE Int. Symp. Microarchitecture*, 2007, pp. 172–182.
[40] H. Zheng, K. Wang, and A. Louri, "A versatile and flexible chiplet-based system design for heterogeneous manycore architectures," in *Proc. 57th ACM/IEEE Des. Automat. Conf.*, 2020, pp. 1–6.
[41] B. Grot, J. Hestness, S. W. Keckler, and O. Mutlu, "Express cube topologies for on-chip interconnects," in *Proc. IEEE Int. Symp. High Perform. Comput. Architecture*, 2009, pp. 163–174.
[42] A. K. Mishra, O. Mutlu, and C. R. Das, "A heterogeneous multiple network-on-chip design: An application-aware approach," in *Proc. ACM/EDAC/IEEE Des. Autom. Conf.*, 2013, pp. 1–10.
[43] H. Zheng, K. Wang, and A. Louri, "Adapt-NoC: A flexible network-on-chip design for heterogeneous manycore architectures," in *Proc. Int. Symp. High- Perform. Comput. Archit.*, 2021, pp. 723–735.
[44] X. Yang et al., "Interstellar: Using halide's scheduling language to analyze DNN accelerators," in *Proc. ACM Int. Conf. Archit. Support Program. Lang. Operating Syst.*, 2020, pp. 369–383.
[45] J. Yang, H. Zheng, and A. Louri, "Venus: A versatile deep neural network accelerator architecture design for multiple applications," in *Proc. 60th ACM/IEEE Des. Automat. Conf.*, San Francisco, CA, USA, 2023, pp. 1–6, doi: 10.1109/DAC56929.2023.10247897.
[46] L. Yin, J. Wang, and H. Zheng, "Exploring architecture, dataflow, and sparsity for GCN accelerators: A holistic framework," in *Proc. Great Lakes Symp. VLSI*, 2023, pp. 489–495.
[47] Y. Shen, M. Ferdman, and P. Milder, "Maximizing CNN accelerator efficiency through resource partitioning," in *Proc. IEEE Int. Symp. Comput. Archit.*, 2017, pp. 535–547.

Jiaqi Yang (Member, IEEE) received the BS degree in electrical engineering and telecommunications engineering with management from the Beijing University of Posts and Telecommunications, China, in 2018, and the MS degree in electrical engineering from George Washington University, Washington, DC, in 2020. Currently, She is working toward the PhD degree in computer engineering with the School of Engineering and Applied Science, George Washington University, Washington, DC. Her research interests primarily lie in computer architecture, Network-on-Chip (NoC), and the design of high-performance, energy-efficient re-configurable DNN accelerators.

Hao Zheng (Member, IEEE) received the BS degree in electrical engineering from Beijing Jiaotong University, Beijing, China, and the PhD degree in computer engineering from George Washington University, Washington, DC. He is currently an assistant professor of electrical and computer engineering with the University of Central Florida, Orlando, Florida. His research interests include computer architecture and parallel computing, with emphasis on interconnection networks, machine learning techniques for efficient computing, and energy-efficient manycore architecture designs.

Ahmed Louri (Fellow, IEEE) received the PhD degree in computer engineering from the University of Southern California, Los Angeles, California, in 1988. He is the David and Marilyn Karlgaard Endowed chair professor of electrical and computer engineering with the George Washington University, Washington, DC, which he joined in 2015. He is also the director of High Performance Computing Architectures and Technologies Laboratory. From 1988 to 2015, he was a professor of electrical and computer engineering with the University of Arizona, Tucson, Arizona, and during that time, he served six years (2000 to 2006) as the chair of the Computer Engineering Program. From 2010 to 2013, he served as a program director with the National Science Foundation's (NSF) Directorate for Computer and Information Science and Engineering. He directed the core computer architecture program and was on the management team of several cross-cutting programs. He conducts research in the broad area of computer architecture and parallel computing, with emphasis on interconnection networks, optical interconnects for parallel computing systems, reconfigurable computing systems, and power-efficient and reliable Network-on-Chips (NoCs) for multicore architectures. Recently he has been concentrating on energy-efficient, reliable, and high-performance many-core architectures, accelerator-rich reconfigurable heterogeneous architectures, machine learning techniques for efficient computing, memory, and interconnect systems, emerging interconnect technologies (photonic, wireless, RF, hybrid) for NoCs, future parallel computing models and architectures (including convolutional neural networks, deep neural networks, and approximate computing), and cloud-computing and data centers. He is the recipient of 2020 IEEE Computer Society Edward J. McCluskey Technical Achievement Award for pioneering contributions to the solution of on-chip and off-chip communication problems for parallel computing and many-core architectures. He is currently the editor-in-chief of *IEEE Transactions on Computers*.