A High-Performance and Energy-Efficient Photonic Architecture for Multi-DNN Acceleration

Yuan Li, *Member, IEEE*, Ahmed Louri, *Fellow, IEEE*, and Avinash Karanth, *Senior Member, IEEE*

***Abstract*—Large-scale deep neural network (DNN) accelerators are poised to facilitate the concurrent processing of diverse DNNs, imposing demanding challenges on the interconnection fabric. These challenges encompass overcoming performance degradation and energy increase associated with system scaling while also necessitating flexibility to support dynamic partitioning and adaptable organization of compute resources. Nevertheless, conventional metallic-based interconnects frequently confront inherent limitations in scalability and flexibility. In this paper, we leverage silicon photonic interconnects and adopt an algorithm-architecture co-design approach to develop MDA, a DNN accelerator meticulously crafted to empower high-performance and energy-efficient concurrent processing of diverse DNNs. Specifically, MDA consists of three novel components: 1) a resource allocation algorithm that assigns compute resources to concurrent DNNs based on their computational demands and priorities; 2) a dataflow selection algorithm that determines off-chip and on-chip dataflows for each DNN, with the objectives of minimizing off-chip and on-chip memory accesses, respectively; 3) a flexible silicon photonic network that can be dynamically segmented into sub-networks, each interconnecting the assigned compute resources of a certain DNN while adapting to the communication patterns dictated by the selected on-chip dataflow. Simulation results show that the proposed MDA accelerator outperforms other state-of-the-art multi-DNN accelerators, including PREMA, AI-MT, Planaria, and HDA. MDA accelerator achieves a speedup of 3.6, accompanied by substantial improvements of 7.3×, 12.7×, and 9.2× in energy efficiency, service-level agreement (SLA) satisfaction rate, and fairness, respectively.**

***Index Terms*—Accelerator, dataflow, deep neural network, silicon photonics.**

I. Introduction

THE proliferation of deep neural network (DNN) accelerators within the realm of cloud computing has emerged as a prominent trend [1], [2], [3], [4] driven by an overarching goal of facilitating concurrent processing of various DNNs. It imposes stringent demand on the underlying interconnection fabric [5], necessitating not only the support for addressing challenges in latency, bandwidth, and energy stemming from system scaling but also the provision of connection flexibility required for active partitioning and adaptable organization of compute resources. However, conventional metallic-based interconnects confront increasingly pronounced scaling limitations [6] as well as inherent rigidity [7], [8], [9], rendering them inadequate to facilitate high-performance and energy-efficient concurrent processing of diverse DNNs. For instance, some previous accelerators [10], [11], [12], [13] facilitate the concurrent processing of multiple DNNs but are confined to a fixed dataflow [4]. Conversely, other accelerators [14], [15], [16] accommodate multiple dataflows but are only capable of concurrently processing a limited number of DNNs.

Silicon photonic interconnects [17], [18] offer several well-established advantages when compared to the metallic-based counterparts, such as distance-independent latency [17], high bandwidth density [18], and high energy efficiency, making them a compelling alternative to architect the interconnection fabric for large-scale DNN accelerators [7], [19]. Moreover, due to their unique characteristics in modulation, transmission, multiplexing, and filtering [17], silicon photonic interconnects exhibit remarkable inherent flexibility which enables efficient support for dynamic resource partitioning to minimize the communication interference between concurrently processed DNNs, as well as adaptable resource organization to accommodate the diversity in DNN characteristics and dataflows.

We introduce the MDA accelerator in this paper, which is specifically designed to enhance the performance and energy efficiency of Multi-DNN Acceleration. MDA is optimized via an algorithm-architecture co-design approach and incorporates three components: 1) a resource allocation algorithm that assigns compute resources to concurrent DNNs based on their computational demands and priorities; 2) a dataflow selection algorithm that sequentially determines off-chip and on-chip dataflows for a DNN to minimize the incurred off-chip and on-chip memory accesses, respectively; 3) a flexible silicon photonic network that can be dynamically segmented into sub-networks, each interconnecting compute resources assigned to a certain DNN while adapting to communication patterns dictated by the selected on-chip dataflow. Simulation results show that the proposed MDA accelerator outperforms other state-of-the-art multi-DNN accelerators [10], [11], [12], [15], achieving evident speedup and improvements in energy efficiency, service-level agreement (SLA) satisfaction rate, and fairness. The significant contributions of this paper include:

- *Resource Allocation Algorithm:* The proposed MDA resource allocation algorithm assigns compute resources to

Manuscript received 27 April 2023; revised 11 October 2023; accepted 22 October 2023. Date of publication 25 October 2023; date of current version 20 November 2023. This work was supported by the National Science Foundation under Grants CCF-1702980, CCF-1812495, CCF-1901165, CCF-1953980, CCF-1513606, CCF-1703013, and CCF-1901192. Recommended for acceptance by Y. Yang. *(Corresponding author: Yuan Li.)*

Yuan Li and Ahmed Louri are with the Department of Electrical and Computer Engineering, George Washington University, Washington, DC 20052 USA (e-mail: liyuan5859@gwu.edu; louri@gwu.edu).

Avinash Karanth is with the School of Electrical Engineering and Computer Science, Ohio University, Athens, OH 45701 USA (e-mail: karanth@ohio.edu).

Digital Object Identifier 10.1109/TPDS.2023.3327535

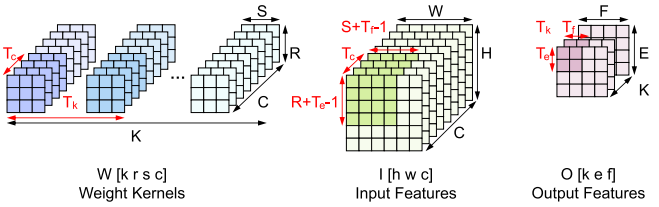


Fig. 1. Computation in $\langle K : E : F : C : R : S \rangle$ convolutional layer and a processing tile $\langle T_k : T_e : T_f : T_c : R : S \rangle$.

concurrently processed DNNs by considering two pivotal factors: the remaining multiple-accumulate (MAC) operations and the time left until the predefined completion deadline. The factors encapsulate the computational demand and priority of a DNN. The output of this resource allocation algorithm serves as input for the next dataflow selection algorithm.

- *Dataflow Selection Algorithm:* We undertake an exhaustive search for dataflow design space and establish quantitative models to precisely quantify the relation between dataflow configuration and memory access count both in the off-chip and on-chip cases. The proposed MDA dataflow selection algorithm harnesses these models in conjunction with the output of the MDA resource allocation algorithm and the parameters specific to a given DNN, to ascertain both off-chip and on-chip dataflows that result in minimum off-chip and on-chip memory accesses, respectively. Moreover, the exhaustive search yields a set of communication patterns that must be accommodated by the flexible silicon photonic network.
- *Flexible Silicon Photonic Network:* We design a novel versatile silicon photonic network that can be actively segmented to sub-networks, each interconnecting compute resources assigned to a certain DNN to minimize the communication interference while being adaptively configured into a set of working modes to adequately facilitate the communication patterns dictated by the selected on-chip dataflow.

II. Background

A. DNN Computation

The computation involved in a typical convolutional layer of a DNN is described in Fig. 1 and represented as a nested loop over weight kernels, input features, and output features in Algorithm 1. This nested loop includes iterations on six dimensions, namely the number of input feature channels $\langle c \rangle$, the number of output feature channels $\langle k \rangle$, the height $\langle r \rangle$ and width $\langle s \rangle$ of weight kernels, and the height $\langle e \rangle$ and width $\langle f \rangle$ of output feature channels. It should be noted that the height and width of input feature channels are not independent and can be derived from other dimensions. With the assumption of stride $Str = 1$ and batch size $N = 1$, we can identify the reuse opportunities of input features along the $\langle k \rangle$ dimension, as well as the reuse opportunities of weights or input features along both $\langle e \rangle$ and $\langle f \rangle$

Algorithm 1: Computation of a Convolutional Layer.

```
1: for (k = 0; k < K; k += 1)
2:   for (e = 0; e < E; e += 1)
3:     for (f = 0; f < F; f += 1)
4:       for (c = 0; c < C; c += 1)
5:         for (r = 0; r < R; r += 1)
6:           for (s = 0; s < S; s += 1)
7:             O[k e f] +=
                 I[r + e − 1 s + f − 1 c] × W[k r s c]
```

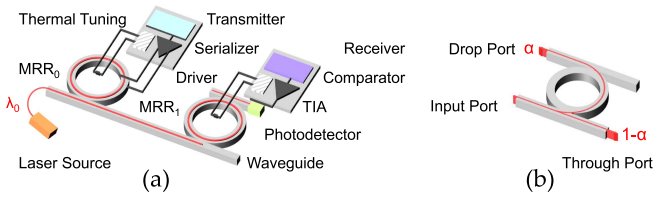


Fig. 2. (a) A silicon photonic interconnect which connects a transmitter and a receiver and (b) an MRR with a resonant wavelength of λ_0 and a split ratio of $\alpha/(1-\alpha)$.

dimensions, following the taxonomy outlined in [20]. We focus on accelerating the convolutional layers, the fully-connected layers, as well as the depth-wise separable convolutional layers as they constitute a significant fraction of all layers in typical DNNs [21], [22].

B. Silicon Photonic Interconnects

We demonstrate the architecture of a typical simple silicon photonic interconnect in Fig. 2. Light in a certain wavelength λ_0 is coupled from an off-chip laser to an on-chip waveguide and traverses from a transmitter to a receiver. On transmitter end, electrical data is serialized and utilized as a modulation signal to modulate the microring resonator (MRR) labeled as MRR_0. On receiver end, the MRR labeled as MRR_1 collects the light in wavelength λ_0 and forwards it to a photodetector to generate a photocurrent signal, which is then amplified by a transimpedance amplifier (TIA) and sent to a comparator to retrieve the original electrical data. A communication channel is established when MRR_0 and MRR_1 in this example share a resonant wavelength λ_0. A resistive heater controlled by a thermal tuning module is attached to every MRR to mitigate both thermal and process variations. The latency and power consumption of this communication channel depend on the transmitter and receiver, and are largely independent of the physical distance between them [17]. The bandwidth density of this communication channel can be easily scaled by adding additional transmitters and receivers with distinct resonant wavelengths other than λ_0, known as wavelength-division multiplexing (WDM). When adding additional receivers that possess λ_0 as resonant wavelength, a single-write-multiple-read (SWMR) multicast channel is created. It achieves higher power efficiency compared to the original unicast channel by sharing the transmitter at the cost of a moderate increase in laser power. The proposed MDA accelerator leverages the advantages of silicon photonic

interconnects to overcome the challenges stemming from system scaling.

The resonant wavelength of an MRR could be thermally tuned [23], [24]. Recent work has reported tuning latency and power as low as 50 ns and 2.06 nm/mW [24], respectively. The split ratio (ratio of light observed in drop and through ports shown in Fig. 2) of an MRR could be electrically tuned [25], [26]. Recent work has reported tuning latency as low as 500 ps [25]. The flexible silicon photonic network inside the proposed MDA accelerator exploits thermal and electrical tuning approaches to construct a variety of working modes to adapt to different communication patterns.

III. MDA Resource Allocation

Algorithm 2 describes how the proposed MDA resource allocation and dataflow selection algorithms are sequentially performed. TheTASKUPDATE ($Task$, $alloc$) function oversees the working status of the MDA accelerator by periodically updating $Task$. The MDA resource allocation algorithm is activated by setting $alloc$ upon arrival of new or completion of existing DNNs. In the case where a new DNN arrives, its number of MAC operations $W_{overall}$, predefined completion deadline T_{target}, estimated execution time when executed on the MDA accelerator alone without interruptions $T_{isolate}$, and parameters from each layer are extracted from the offline information, before $alloc$ is set. Similarly, in the case where an existing DNN is completed, the corresponding entry in $Task$ is deleted while its shares of compute resources M^i and on-chip memory O^i are released before $alloc$ is set. In either case above, the output of the previous invocation of the MDA resource allocation algorithm no longer reflects the latest computational demands and priorities of the concurrently processed DNNs, calling for a new invocation of the MDA resource allocation algorithm. During operation without the arrival of new or completion of existing DNNs, the estimated remaining execution time T_{remain} and the remaining time until the predefined completion deadline $T_{deadline}$, which are defined in Equation (1) and (2), respectively, are regularly updated for each concurrently processed DNN. Please note that W_{remain} and $T_{current}$ represent the number of remaining MAC operations and current time, respectively.

$$T^i_{remain} = T^i_{isolate} \times W^i_{remain}/W^i_{overall} \tag{1}$$

$$T^i_{deadline} = T^i_{target} - T_{current} \tag{2}$$

The MDA resource allocation algorithm assigns compute resources to concurrently processed DNNs following Equation (3). M and M^i represent the overall compute resources and the share assigned to a DNN, respectively. M^i is linearly proportional to the computational demand represented by T_{remain} and exponentially proportional to priority estimated by $T_{deadline}$. Such exponential proportion significantly favors DNNs with high priorities (small $T_{deadline}$ values), promoting equal progress of concurrently processed DNNs while ensuring adherence to the predefined completion deadlines. Moreover, it helps assign proper compute resources to DNNs that have missed their predefined completion deadlines (with negative $T_{deadline}$ values). The share of on-chip memory O^i is proportionally assigned based on M^i. The proposed MDA dataflow selection algorithm is performed upon completion of the MDA resource allocation algorithm, as the updated assignment of compute resources and on-chip memory can potentially make current dataflow decisions non-optimal.

Algorithm 2: MDA Algorithms.

```
function MDAALGORITHMS (Task)                    ▷ Task: multiplexed DNNs
  for time ← [ 0, ∞) do                          ▷ Update the progress of each DNN
    TASKUPDATE (Task, alloc)                     ▷ alloc: set for new/completed DNN
      if alloc = true then
        TASKALLOCATION (Task)                    ▷ Resource allocation
        TASKDATAFLOW (Task)                      ▷ Dataflow selection
        alloc = false                            ▷ Clear alloc
      end if
    time +=1
  end for
end function
```

$$M^i = M \times \frac{T^i_{remain} \times e^{-T^i_{deadline}}}{\sum T^i_{remain} \times e^{-T^i_{deadline}}} \tag{3}$$

IV. MDA Dataflow Selection

We take a two-step approach to sequentially select the off-chip and on-chip dataflows for each concurrently processed DNN, with the objectives of minimizing off-chip and on-chip memory accesses, respectively. Please note that this approach does not necessarily yield optimal dataflow decisions as off-chip and on-chip dataflows will interact with each other, and there is no clear definition of the global optimization target. The proposed two-step approach may potentially lead to sub-optimal dataflow decisions, however, significantly reduces the search space.

A. Off-Chip Dataflow Selection

A DNN accelerator typically consists of off-chip memory (e.g., DRAM), on-chip memory (e.g., global buffer (GLB)), and processing elements (PEs) [27]. Fig. 1 illustrates how a convolutional layer $\langle K : E : F : C : R : S\rangle$ can be partitioned into tiles with tile size $\langle T_k : T_e : T_f : T_c : R : S\rangle$. The tiling process naturally divides the overall dataflow into two parts: off-chip dataflow which orchestrates data movements between off-chip memory and on-chip memory, and on-chip dataflow which orchestrates data movements between on-chip memory and numerous PEs. Please note that we assume a unified on-chip memory shared by all PEs as in [27] in this work. The off-chip dataflow optimization goal is to minimize the off-chip memory accesses as they are notably more costly than the actual MAC operations [20].

TABLE I
OFF-CHIP SECTION DATAFLOW EXPLORATION

Reuse	Data	On-Chip Memory Share	Off-Chip Memory Access
Weight	W	$T_k \times T_c \times R \times S$	$K \times C \times R \times S$
	I	$T_c \times (R+T_e-1) \times (S+T_f-1)$	$K \times C \times E \times F \times (R+T_e-1) \times (S+T_f-1)/T_k \times T_e \times T_f$
	P	$T_k \times T_e \times T_f$	$2 \times K \times C \times E \times F/T_c$
Input	W	$T_k \times T_c \times R \times S$	$K \times C \times E \times F \times R \times S/T_e \times T_f$
	I	$T_c \times (R+T_e-1) \times (S+T_f-1)$	$C \times E \times F \times (R+T_e-1) \times (S+T_f-1)/T_e \times T_f$
	P	$T_k \times T_e \times T_f$	$2 \times K \times C \times E \times F/T_c$
Output	W	$T_k \times T_c \times R \times S$	$K \times C \times E \times F \times R \times S/T_e \times T_f$
	I	$T_c \times (R+T_e-1) \times (S+T_f-1)$	$K \times C \times E \times F \times (R+T_e-1) \times (S+T_f-1)/T_k \times T_e \times T_f$
	P	$T_k \times T_e \times T_f$	$K \times E \times F$

The input of the off-chip dataflow selection algorithm is the share of on-chip memory for a DNN O^i and the parameters of a certain layer in that DNN $\langle K:E:F:C:R:S\rangle$. The output of the off-chip dataflow selection algorithm includes the proper tile size represented by $\langle T_k:T_e:T_f:T_c:R:S\rangle$ and the tile processing order. Note that we only perform tiling on $\langle k\rangle$, $\langle e\rangle$, $\langle f\rangle$, and $\langle c\rangle$ dimensions to avoid breaking individual weight kernel input channels. A tile with a tile size of $\langle T_k:T_e:T_f:T_c:R:S\rangle$ needs to store $T_k \times T_c \times R \times S$ weights, $T_c \times (R+T_e-1) \times (S+T_f-1)$ input features, and $T_k \times T_e \times T_f$ partial sums in the on-chip memory, which must be fully accommodated by the assigned share O^i. For a given tile size, three processing orders that maximize the reuse of partial sums, input features, and weights could be derived by processing the $\langle c\rangle$, $\langle k\rangle$, and $\langle e\ f\rangle$ dimensions in the innermost loops. Algorithm 3 shows an example of an off-chip dataflow that maximizes the reuse of partial sums.

We conduct an exhaustive search for the off-chip memory access count when varying the tile size and tile processing order and list the quantitative model in Table I. The overall off-chip memory access count values for three different tile processing orders V_{weight}, V_{input}, and V_{output} for a given tile size $\langle T_k:T_e:T_f:T_c:R:S\rangle$ can be obtained by adding up the corresponding entries in the last column of Table I. The right combination of tile size and tile processing order that yields minimum off-chip memory accesses is considered the optimal off-chip dataflow, which is defined in Equation (4). An optimal off-chip dataflow for a DNN shall fully leverage the assigned on-chip memory share O^i while maximizing data reuse between the successively processed tiles.

$$V^i = min(V^i_{weight}, V^i_{input}, V^i_{output})$$
$$\text{s.t. } O_{<T_k:T_e:T_f:T_c:R:S>} \leq O^i \tag{4}$$

Algorithm 3: Off-Chip Section Dataflow Example.

```
for (t_k = 0; t_k < K; t_k += T_k)                    ▷ Off-chip section
  for (t_e = 0; t_e < E; t_e += T_e)
    for (t_f = 0; t_f < F; t_f += T_f)
      for (t_c = 0; t_c < C; t_c += T_c)
        for (k = t_k; k < min (K, t_k + T_k); k += 1)   ▷ On-chip section
          for (e = t_e; e < min (E, t_e + T_e); e += 1)
            for (f = t_f; f < min (F, t_f + T_f); f += 1)
              for (c = t_c; c < min (C, t_c + T_c); c += 1)
                for (r = 0; r < R; r += 1)
                  for (s = 0; s < S; s += 1)
                    O[k e f] += I[r + e − 1 s + f − 1 c] × W[k r s c]
```

Algorithm 4: On-Chip Section Dataflow Example.

```
parallel_for (p_k = t_k; p_k < min (K, t_k + T_k); p_k += ⌈T_k/P_k⌉)
  parallel_for (p_e = t_e; p_e < min (E, t_e + T_e); p_e += ⌈T_e/P_e⌉)
    parallel_for (p_f = t_f; p_f < min (F, t_f + T_f); p_f += ⌈T_f/P_f⌉)
      for (k = p_k; k < min (K, t_k + T_k, p_k + ⌈T_k/P_k⌉); k += 1)
        for (e = p_e; e < min (E, t_e + T_e, p_e + ⌈T_e/P_e⌉); e += 1)
          for (f = p_f; f < min (F, t_f + T_f, p_f + ⌈T_f/P_f⌉); f += 1)
            for (c = t_c; c < min (C, t_c + T_c); c += 1)
              for (r = 0; r < R; r += 1)
                for (s = 0; s < S; s += 1)
                  O[k e f] += I[r + e − 1 s + f − 1 c] × W[k r s c]
```

B. On-Chip Dataflow Selection

The goal of on-chip dataflow optimization is to minimize the on-chip memory accesses which are also more costly than MAC operations [20]. Unlike off-chip dataflows which likely incur only temporal data reuse, on-chip dataflows can result in both temporal and spatial data reuse occasions.

The input of the on-chip dataflow selection algorithm is the share of compute resources for a DNN M^i as well as the tile parameters of a certain layer $\langle T_k:T_e:T_f:T_c:R:S\rangle$ in that DNN. Please note that M^i is measured by the number of PEs, or PE partitions if allocation of compute resources is performed at a coarser granularity. The output of the on-chip dataflow selection algorithm is the right spatial and temporal distribution of MAC operations across PEs and within each PE, respectively. Algorithm 4 shows an example of an on-chip dataflow that processes the MAC operations in parallel in $P_k \times (P_e \times P_f)$ PEs while maximizing reuse of partial sums within each PE similar to [19], [28]. $P_k \times (P_e \times P_f)$ must not exceed the assigned compute resources M^i. An optimal on-chip dataflow for a DNN tile shall fully leverage the spatial parallelism capability

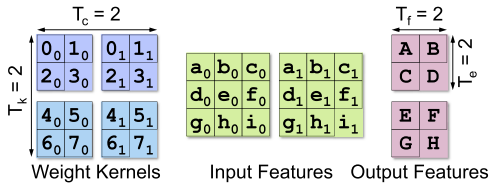


Fig. 3. Tile example to illustrate on-chip dataflow selection.

TABLE II
COMMUNICATION PATTERNS

Dimension	Data	Spatial	Temporal	Communication Patterns
$\langle k \rangle$	W	✘	✔	GLB → PE Unicast
	I	✔	✔	GLB → PE Multicast
	P		✔	GLB ↔ PE Unicast
$\langle c \rangle$	W	✘	✔	GLB → PE Unicast
	I	✘	✔	GLB → PE Unicast
	P		✔	GLB ↔ PE PE → PE Unicast
$\langle e\ f \rangle$	W	✔	✘	GLB → PE Multicast
	I	✔	✔	GLB → PE Multicast
	P		✔	GLB ↔ PE Unicast
$\langle r\ s \rangle$	W	✘	✔	GLB → PE Unicast
	I	✔	✔	GLB → PE Multicast
	P		✔	GLB ↔ PE PE → PE Unicast
$\langle k\ e\ f \rangle$	W	✔	✘	GLB → PE Multicast
	I	✔	✘	GLB → PE Multicast
	P		✔	GLB ↔ PE Unicast
$\langle k\ e\ f\ c \rangle$	W	✔	✘	GLB → PE Multicast
	I	✔	✘	GLB → PE Multicast
	P		✔	GLB ↔ PE PE → PE Unicast
$\langle k\ e\ f\ c\ r\ s \rangle$	W	✔	✘	GLB → PE Multicast
	I	✔	✘	GLB → PE Multicast
	P		✘	GLB ↔ PE PE → PE Unicast

provided by the numerous PEs while maximizing temporal data reuse inside each individual PE.

1) On-Chip Dataflow Exploration: We study the data reuse opportunities of different on-chip dataflows on tile in Fig. 3 and summarize the resulting set of communication patterns that must be accommodated by the flexible silicon photonic network in the proposed MDA accelerator in Table II. We identify four communication patterns: GLB → PE unicast, GLB ← PE unicast, PE → PE unicast, and GLB → PE multicast. In cases like processing MAC operations in parallel in $\langle e\ f \rangle$ dimensions, transmissions of weights and input features incur the GLB → PE multicast communication pattern. As a result, the flexible silicon photonic network in the proposed MDA accelerator must support simultaneous multicast from GLB to two orthogonal PE sets.

No Parallelism: Fig. 4(a) lists the MAC operations involved in processing the DNN tile shown in Fig. 3. Only PE_0 is used. There are four types of temporal data reuse occasions: weight reuse due to convolutional sliding window (e.g., weight 0_0 in $0_0 \times a_0$, $0_0 \times b_0$, $0_0 \times d_0$, and $0_0 \times e_0$), input feature reuse due to convolutional sliding window (e.g., input feature e_0 in $3_0 \times e_0$, $2_0 \times e_0$, $1_0 \times e_0$, and $0_0 \times e_0$), input feature reuse due to multiple weight kernels (e.g., input feature a_0 in $0_0 \times a_0$ and $4_0 \times a_0$), and partial sum reuse in generating any output feature. Please note that the weight and input feature reuse occasions due to convolutional sliding window do not occur simultaneously because of the pairwise operation nature. The temporal data reuse occasions can be leveraged by adjusting the MAC operations inside PE_0. For instance, sequentially processing $0_0 \times a_0$, $0_0 \times b_0$, $0_0 \times d_0$, and $0_0 \times e_0$ leverages the weight reuse opportunities while sacrificing the input feature and partial sum reuse opportunities. GLB → PE unicast and GLB ← PE unicast communication patterns are involved.

Parallelism in $\langle k \rangle$ Dimension: Fig. 4(b) demonstrates the parallel distribution of MAC operations in the $\langle k \rangle$ dimension. PE_0 and PE_1 are used as $T_k = 2$. We observe that temporal input feature reuse due to multiple weight kernels is turned into spatial reuse (e.g., input feature a_0 in $0_0 \times a_0$ on PE_0 and $4_0 \times a_0$ on PE_1) while the other three temporal data reuse occasions remain unchanged. Consequently, one additional communication pattern, GLB → PE multicast, is involved in potential input feature transmission.

Parallelism in $\langle c \rangle$ Dimension: Fig. 4(c) demonstrates the parallel distribution of MAC operations in the $\langle c \rangle$ dimension. PE_0 and PE_1 are used as $T_c = 2$. We observe that temporal partial sum reuse is undermined in part while the other three temporal data reuse occasions remain unchanged. The partial sums generated from PE_0 and PE_1 must be accumulated to generate any output feature. One additional communication pattern, PE → PE unicast, is accordingly involved in potential partial sum transmission.

Parallelism in $\langle e\ f \rangle$ Dimensions: Fig. 4(d) illustrates the parallel distribution of MAC operations in both $\langle e \rangle$ and $\langle f \rangle$ dimensions. Four PEs, PE_0 to PE_3, are used as $T_e \times T_f = 4$. We observe that temporal weight and input feature reuse occasions due to convolutional sliding window are turned into spatial reuse occasions (e.g., weight 0_0 in $0_0 \times a_0$, $0_0 \times b_0$, $0_0 \times d_0$, and $0_0 \times e_0$, input feature e_0 in $3_0 \times e_0$, $2_0 \times e_0$, $1_0 \times e_0$, and $0_0 \times e_0$) while the other two temporal data reuse occasions remain unchanged. Note that the generated spatial weight and input feature reuse occasions still do not occur simultaneously because of the pairwise operation nature. Consequently, the GLB → PE multicast communication pattern is involved in the potential transmission of either weight or input feature data type.

Parallelism in $\langle r\ s \rangle$ Dimensions: Fig. 4(e) illustrates the parallel distribution of MAC operations in both $\langle r \rangle$ and $\langle s \rangle$ dimensions. Four PEs, PE_0 to PE_3, are used since $R \times S = 4$. We observe that temporal partial sum reuse is undermined in part while temporal input feature reuse due to convolutional sliding window is turned into spatial reuse (e.g., input feature e_0 in $3_0 \times e_0$, $2_0 \times e_0$, $1_0 \times e_0$, and $0_0 \times e_0$). Meanwhile, another two temporal data reuse occasions remain unchanged. As a result, the additional GLB → PE multicast and PE → PE unicast communication patterns are involved to potentially transmit input feature and partial sum data types, respectively.

Parallelism in $\langle k\ e\ f \rangle$ Dimensions: Fig. 4(f) illustrates the parallel distribution of MAC operations in $\langle k \rangle$, $\langle e \rangle$, and $\langle f \rangle$ dimensions. Eight PEs, PE_0 to PE_7, are used as $T_k \times T_e \times T_f = 8$. We observe that only the temporal reuse of partial sum remains unchanged. Temporal weight and input feature reuse occasions due to convolutional sliding window (e.g., weight 0_0 in $0_0 \times a_0$, $0_0 \times b_0$, $0_0 \times d_0$, and $0_0 \times e_0$, input feature e_0 in $3_0 \times e_0$, $2_0 \times e_0$, $1_0 \times e_0$, and $0_0 \times e_0$), as well as temporal

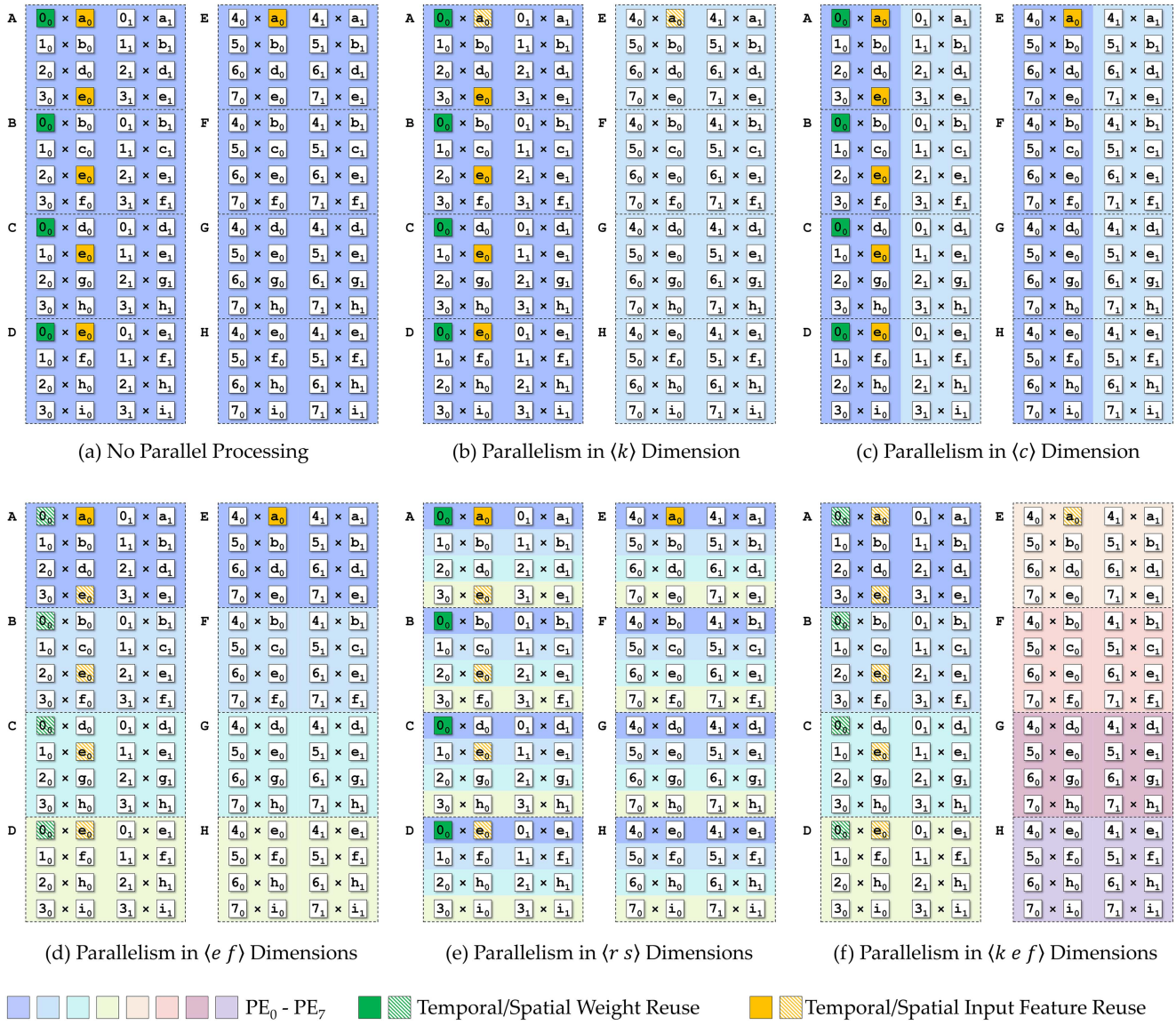


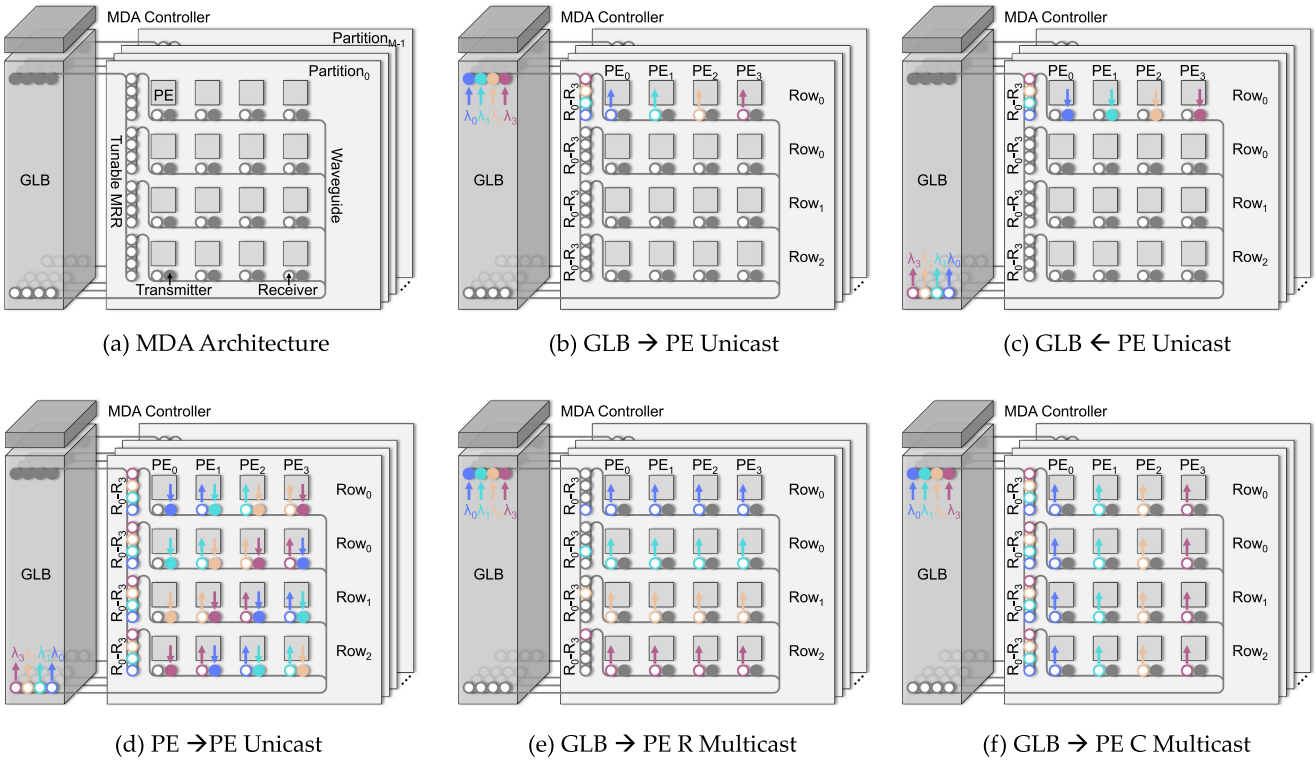
Fig. 4. Temporal and spatial distribution of MAC operations across PEs and within each PE and the consequent data reuse instances. We use the DNN tile $\langle T_k : T_e : T_f : T_c : R : S \rangle = \langle 2 : 2 : 2 : 2 : 2 : 2 \rangle$ as an example.

input feature reuse due to multiple weight kernels (e.g., input feature a_0 in $0_0 \times a_0$ on PE_0 and $4_0 \times a_0$ on PE_1), are turned into spatial reuse occasions. Hence, the GLB $\rightarrow$ PE multicast communication pattern is involved in the potential transmission of both weight and input feature data types.

Parallelism in $\langle k\ e\ f\ c \rangle$ Dimensions: If compared against the above case of distributing MAC operations in parallel in $\langle k \rangle$, $\langle e \rangle$, and $\langle f \rangle$ dimensions, increasing parallel distribution in the additional $\langle c \rangle$ dimension does not incur any conversions of temporal data reuse opportunities into spatial data reuse opportunities while the temporal partial sum reuse is somehow partially undermined. Consequently, distributing MAC operations in parallel in the additional $\langle c \rangle$ dimension is only beneficial when the benefit from doubling the involved PEs ($T_c = 2$) can mitigate the drawback due to the partial loss of temporal data locality.

Parallelism in $\langle k\ e\ f\ c\ r\ s \rangle$ Dimensions: This represents an extreme situation where MAC operations are processed in parallel in every possible dimension while temporal locality is completely lost. The optimal on-chip dataflow lies between the two extremes of not having computation parallelism in any dimension, and having parallelism in all dimensions.

2) Key Observations: We make the following five observations according to the on-chip dataflow exploration from the perspective of parallel processing: 1) increasing processing parallelism among PEs leads to the loss of temporal data reuse opportunities, which, however, are converted into spatial data reuse opportunities that can be leveraged by the efficient multicast capability of silicon photonics; 2) parallel processing in the $\langle k \rangle$, $\langle e \rangle$, and $\langle f \rangle$ dimensions is most beneficial as the lost temporal data reuse opportunities are entirely converted



(a) MDA Architecture (b) GLB → PE Unicast (c) GLB ← PE Unicast

(d) PE →PE Unicast (e) GLB → PE R Multicast (f) GLB → PE C Multicast

Fig. 5. MDA architecture overview and configurations for different working modes.

into spatial data reuse opportunities; 3) parallel processing in the $\langle r \rangle$ and $\langle s \rangle$ dimensions is less beneficial because the same spatial input feature reuse opportunities can be obtained through parallel processing in the $\langle e \rangle$ and $\langle f \rangle$ dimensions while extra fraction of the temporal partial sum reuse opportunities is lost; 4) parallel processing in the $\langle c \rangle$ dimension is least beneficial as it only leads to the lost of a fraction of the temporal partial sum reuse opportunities; 5) parallel processing in all three $\langle k \rangle$, $\langle e \rangle$, and $\langle f \rangle$ dimensions represents an ideal case where high parallelism and full usage of the data reuse opportunities are achieved simultaneously [19].

V. MDA Accelerator Architecture

A. Architecture Overview

Fig. 5(a) shows an architectural overview of the proposed MDA accelerator. It includes a GLB and M partitions, where each partition contains $N \times N$ ($N = 4$ in Fig. 5 for clarity) PEs. Each partition is connected to the GLB through a dedicated waveguide with N transmitters and N receivers on the GLB end. Each PE in any partition is equipped with a transmitter whose resonant wavelength is thermally tuned and a receiver whose resonant wavelength and split ratio are thermally and electrically tuned, respectively. In front of each row of PEs in a partition, N MRRs with fixed resonant wavelengths and tunable split ratios are implemented to forward a particular fraction of laser power in N wavelengths to PEs in this row. There are in total $M \times N \times (3 \times N + 2)$ MRRs in the MDA accelerator. Please note that each PE performs dot-product along the $\langle c \rangle$ dimension to exploit efficient spatial reduction as in [6]. Besides, we can use the space-division multiplexing technique to deploy multiple waveguides between the GLB and a given partition. The MDA Controller is responsible for undertaking the resource allocation and dataflow selection algorithms described in Algorithm 3 and tuning the resonant wavelength and split ratio of each MRR inside a partition.

B. MDA Silicon Photonic Network Working Modes

The flexible silicon photonic network in the MDA accelerator can be segmented at runtime and adaptively configured into a collection of working modes to efficiently facilitate the communication patterns extracted from the on-chip dataflow exploration. Fig. 5(b)-(f) and Table III describe all five working modes in support of four communication patterns and how they are implemented in one partition. Please note that we utilize a specific color to represent a resonant wavelength. In particular, ✗ and ✓ represent that an MRR is at off-resonant and on-resonant states, respectively. a/b represents the split ratio of an MRR.

GLB → PE Unicast Mode: Fig. 5(b) illustrates the working mode that facilitates the GLB → PE unicast communication pattern. In this working mode, the GLB sequentially communicates with different rows of PEs in a partition due to the mismatch between the number of PEs in a partition $N \times N$ and the number of available wavelengths N. Fig. 5(b) shows

TABLE III
MDA SILICON PHOTONIC NETWORK WORKING MODES

Mode	Row	R_0	R_1	R_2	R_3	PE_0		PE_1		PE_2		PE_3	
						R	T	R	T	R	T	R	T
GLB → PE Unicast	0	1/0	1/0	1/0	1/0	1/0	✗	1/0	✗	1/0	✗	1/0	✗
	1	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
	2	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
	3	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
GLB ← PE Unicast	0	1/0	1/0	1/0	1/0	✗	✔	✗	✔	✗	✔	✗	✔
	1	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
	2	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
	3	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
PE → PE Unicast	0	1/3	1/3	1/3	1/3	✗	✔	1/0	✔	1/0	✔	1/0	✔
	1	1/2	1/2	1/2	1/2	✗	✔	1/0	✔	1/0	✔	1/0	✔
	2	1/1	1/1	1/1	1/1	✗	✔	1/0	✔	1/0	✔	1/0	✔
	3	1/0	1/0	1/0	1/0	✗	✔	1/0	✔	1/0	✔	1/0	✔
GLB → PE R Multicast	0	1/0	✗	✗	✗	1/3	✗	1/2	✗	1/1	✗	1/0	✗
	1	✗	1/0	✗	✗	1/3	✗	1/2	✗	1/1	✗	1/0	✗
	2	✗	✗	1/0	✗	1/3	✗	1/2	✗	1/1	✗	1/0	✗
	3	✗	✗	✗	1/0	1/3	✗	1/2	✗	1/1	✗	1/0	✗
GLB → PE C Multicast	0	1/3	1/3	1/3	1/3	1/0	✗	1/0	✗	1/0	✗	1/0	✗
	1	1/2	1/2	1/2	1/2	1/0	✗	1/0	✗	1/0	✗	1/0	✗
	2	1/1	1/1	1/1	1/1	1/0	✗	1/0	✗	1/0	✗	1/0	✗
	3	1/0	1/0	1/0	1/0	1/0	✗	1/0	✗	1/0	✗	1/0	✗

the case of unicast communication from the GLB to $Row\ _0$ of PEs in a partition. The transmitters on the GLB end work on resonant wavelengths $\lambda_0 - \lambda_3$ to send data. The four tunable MRRs $R_0 - R_3$ also work on resonant wavelengths $\lambda_0 - \lambda_3$ and are electrically tuned to a universal split ratio of $1/0$, forwarding all the laser power of wavelengths $\lambda_0 - \lambda_3$ to PEs in $Row\ _0$. Meanwhile, the receivers connected to $PE_0 - PE_3$ in $Row\ _0$ are thermally tuned to resonant wavelengths $\lambda_0 - \lambda_3$, respectively, and electrically tuned to a universal split ratio of $1/0$. In a sense, all the laser power of any specific wavelength that carries transmitted data is received by a corresponding PE in $Row\ _0$, facilitating the unicast communication from the GLB to PEs in $Row\ _0$. Unicast communications to other rows are subsequently accomplished following a similar approach.

GLB ← PE Unicast Mode: Fig. 5(c) illustrates the working mode that facilitates the GLB ← PE unicast communication pattern. In this working mode, different rows of PEs in a partition sequentially communicate with the GLB due to the mismatch between the number of PEs in a partition $N \times N$ and the number of available wavelengths N. Fig. 5(b) shows the case of unicast communication from $Row\ _0$ of PEs in a partition to the GLB. The four tunable MRRs $R_0 - R_3$ still work on resonant wavelengths $\lambda_0 - \lambda_3$ and are electrically tuned to a universal split ratio of $1/0$, forwarding all the laser power of unmodulated wavelengths $\lambda_0 - \lambda_3$ to PEs in $Row\ _0$. Meanwhile, the transmitters connected to $PE_0 - PE_3$ in $Row\ _0$ are thermally tuned to resonant wavelengths $\lambda_0 - \lambda_3$, respectively, to send data. The receivers on the GLB end work on resonant wavelengths $\lambda_0 - \lambda_3$. Therefore, data sent by any PE in $Row\ _0$ on the associated wavelength is received by the GLB, facilitating the unicast communication from the PEs in $Row\ _0$ to the GLB. Unicast communications from other rows are subsequently accomplished following a similar approach.

PE → PE Unicast Mode: Fig. 5(d) demonstrates the working mode facilitating the PE → PE unicast communication pattern. In this working mode, each PE sends data to its downstream adjacent PE in the same row while the last PE sends data to the GLB. The four tunable MRRs $R_0 - R_3$ work on resonant wavelengths $\lambda_0 - \lambda_3$ but are electrically tuned to different split ratios depending on their physical locations. Specifically, tunable MRRs in $Row\ _0$, $Row\ _1$, $Row\ _2$, $Row\ _3$ are tuned to split ratios $1/3$, $1/2$, $1/1$, and $1/0$, respectively, forwarding a quarter of the laser power of unmodulated wavelengths $\lambda_0 - \lambda_3$ to PEs in each row. In the meantime, the transmitter connected to each PE is thermally tuned to the same resonant wavelength as its downstream adjacent PE in the same row, while the transmitters connected to the last PEs of different rows are thermally tuned to different resonant wavelengths. For instance, the transmitter and receiver connected to PE_0 and PE_1 in $Row\ _0$ share the resonant wavelength λ_0, while the last PEs in $Row\ _0$, $Row\ _1$, $Row\ _2$, and $Row\ _3$ are tuned to resonant wavelengths λ_3, λ_0, λ_1, and λ_2, respectively. The receivers on the GLB end work on resonant wavelengths $\lambda_0 - \lambda_3$. Therefore, a chain of data transmission is constructed between PE_0, PE_1, PE_2, and PE_3 in each row and the GLB, facilitating the inter-PE unicast communication.

GLB → Row-Wise Multicast Mode: Fig. 5(e) demonstrates the working mode facilitating the GLB→ PE row-wise multicast (R multicast) communication pattern. In this working mode, PEs in different rows receive different data sent from the GLB while PEs in each row receive the same data. The transmitters on the GLB end work on resonant wavelengths $\lambda_0 - \lambda_3$ to send data. Only one tunable MRR is activated and electrically tuned to a split ratio of $1/0$ to forward all the laser power of a specific wavelength to a row. For instance, tunable MRR R_0 in $Row\ _0$ work on resonant wavelength λ_0, forwarding all the laser power of wavelength λ_0 to PEs in $Row\ _0$. Meanwhile, the receivers connected to $PE_0 - PE_3$ in $Row\ _0$ are thermally tuned to resonant wavelength λ_0 and electrically tuned to split ratios $1/3$, $1/2$, $1/1$, and $1/0$, respectively, so that each PE in $Row\ _0$ receives a quarter of the laser power of the modulated wavelength λ_0. Therefore, multicast communication from the GLB to PEs in $Row\ _0$ is accomplished. Multicast communications to other rows are accomplished in parallel following a similar approach.

GLB → Column-Wise Multicast Mode: Fig. 5(f) illustrates the working mode that facilitates the GLB→ PE column-wise multicast (C Multicast) communication pattern. PEs in each row receive different data sent from the GLB while PEs in each column (PEs in the same relative physical position but in different rows) receive the same data. The transmitters on the GLB end work on resonant wavelengths $\lambda_0 - \lambda_3$ to send data. Tunable MRRs with the same resonant wavelength in all four rows are electrically tuned to different split ratios. For instance, tunable MRRs labeled as R_0 in $Row\ _0$, $Row\ _1$, $Row\ _2$, and $Row\ _3$ are tuned to split ratios of $1/3$, $1/2$, $1/1$, and $1/0$, respectively, forwarding a quarter of the laser power of wavelength λ_0 to each row. In the meantime, the receivers connected to PEs labeled as PE_0 in four rows are thermally tuned to the resonant wavelength λ_0 and a universal split ratio of $1/0$, so that each PE labeled as PE_0 receives a quarter of the laser power of the modulated wavelength λ_0. Hence, multicast communication from the GLB to PEs in the first column is accomplished. Multicast communications to other columns are accomplished in parallel in a similar approach.

TABLE IV
SIMULATION PARAMETERS

Parameter	Value
Number of PE Partitions M	8
PE Partition Dimension $N \times N$	16×16
PE Dot-Product Width	8
PE Operating Frequency	700 MHz
Data Width	16 bits
GLB Capacity	12 MB
GLB Bandwidth	480 GB/s
Off-Chip Memory	HBM 358 GB/s

VI. Evaluation Methodology

A. Evaluation Setup

We extend the open-source MAESTRO simulator [29] to model the concurrent processing of heterogeneous DNNs to extract the numbers of MAC operations as well as accesses to each memory hierarchy including local registers, shared GLB, and off-chip memory. The obtained numbers are utilized to derive the execution time that covers both computation and communication aspects when taking the inter-DNN interference into account and enforcing bandwidth constraints along metallic-based or silicon photonic interconnects. The energy consumption can also be derived from these numbers and the power model discussed below. Table IV demonstrates the key simulation parameters we have assumed when modeling the MDA accelerator architecture. We have assumed similar parameters as in prior studies [10], [12]. To model the silicon photonic interconnects, we have assumed a bit error rate of 1×10^{-12} at the bit rate of 10 Gbps and wavelengths around 1550 nm. We have also assumed a maximum free spectral range limit of 50 nm [18], reflecting the fabrication limitation of MRR radius. The number of wavelengths utilized in the MDA accelerator architecture is 16. The power penalty due to crosstalk is negligible [30].

B. Power Consumption Model

We extract the power consumption of MAC operations using `Synopsys Design Compiler`. The power consumption of accessing local registers and the on-chip GLB is obtained using `CACTI 6.0` [31] while the power consumption of accessing off-chip memory is obtained using `DRAMSim2` [32]. The energy consumption of a metallic-based interconnect is assumed to be 170 fJ/bit/mm [33] while the power consumption of a silicon photonic interconnect is obtained from Equation (5).

$$P_{overall} = P_{tx} + P_{rx} + P_{laser} + P_{thermal} \tag{5}$$

The overall power consumption ($P_{overall}$) includes four separate parts: the power consumption of transmitters (P_{tx}), the power consumption of receivers (P_{rx}), the power consumption of the off-chip laser source (P_{laser}), and the power consumption of the resistive heaters co-located with MRRs to mitigate both thermal and process variations ($P_{thermal}$). We assume the power consumption of a transmitter per wavelength $P_{tx} = 0.9$ mW and the power consumption of a receiver per wavelength $P_{rx} = 0.6$ mW [34]. We also assume the power consumption of the resistive heater per MRR $P_{thermal} = 0.15$ mW [35]. The power consumption of the off-chip laser source (P_{laser}) is obtained from Equation (6), when assuming the photodetector sensitivity $P_{rs} = -26$ dBm [19], the power penalty due to extinction ratio $P_{extinction} = 2$ dB [36], and the system margin $C_{system} = 4$ dB [37]. Note that the overall insertion loss C_{loss} is obtained by accumulating the insertion loss of each component along a silicon photonic communication channel.

$$P_{laser} = P_{rs} + C_{loss} + P_{extinction} + M_{system} \tag{6}$$

C. Baseline Architectures

The proposed MDA accelerator architecture is compared against other state-of-the-art multi-DNN accelerators including PREMA (PR) [10], AI-MT (AI) [11], Planaria (PL) [12], and HDA [15]. PREMA and AI-MT only support temporal multiplexing of DNNs while Planaria supports temporal and spatial multiplexing of DNNs. HDA supports multiplexing of DNNs on three sub-accelerators with distinct dataflow optimizations. All four baseline architectures as well as the proposed MDA accelerator architecture are set to the same computation capacity for fair comparison. They also share the same GLB size of 12 MB [10], [12], the same data width of 16 bits [27], the same PE clock frequency of 700 MHz, and the same off-chip memory bandwidth of 358 GB/s assuming a high-bandwidth memory (HBM) module.

D. Evaluation Benchmarks

We compile a multi-DNN benchmark from nine DNN models [38], [39], [40], [41], [42], [43], [44] for both image processing and object detection application cases. A randomly arranged sequence of DNNs from the pool of the nine models is applied to the simulation of the proposed MDA accelerator architecture and other baseline architectures. The length of this sequence of DNNs is sufficient to capture diverse DNN multiplexing scenarios. The arrival time of each DNN from this sequence follows a Poisson Distribution as in [12]. The DNN arrival rate λ is measured by the average number of arrived DNNs in one million clock cycles. Each DNN is also assigned an expected completion deadline measured by SLA.

VII. Experiment Results

A. Speedup Results

Speedup measures the relative performance of two architectures working on the same application task. We first fix the DNN arrival rate at $\lambda = 9/M$ while varying the predefined completion deadline SLA from $3 \times T_{isolate}$ to $12 \times T_{isolate}$. A small SLA number indicates an urgent completion deadline that is challenging to fulfill. A large SLA number indicates a loose completion deadline from the application perspective while implying opportunities for better DNN multiplexing setups from the system perspective. The speedup of AI over PR increases insignificantly from 1.17 to 1.29 as AI does not fully leverage the opportunities for better DNN multiplexing setups due to its constraints of only supporting the temporal DNN multiplexing and a fixed dataflow. The speedup of PL over PR increases from 1.07 to 1.62. The

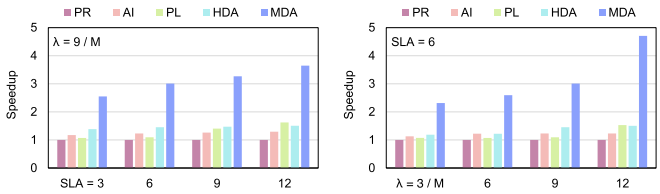


Fig. 6. Speedup compared to the PR baseline when varying SLA (left) and arrival rate λ (right).

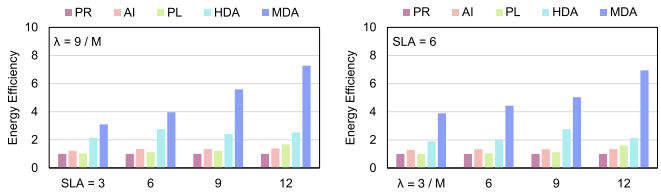


Fig. 7. Energy efficiency comparison when varying SLA (left) and arrival rate λ (right). All values are normalized to the PR baseline.

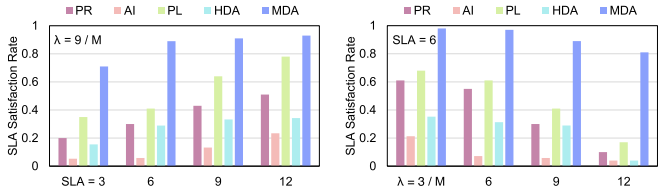


Fig. 8. SLA satisfaction rate comparison when varying SLA (left) and arrival rate λ (right).

increase is trivial in cases with urgent completion deadlines (1.07 for $3 \times T_{isolate}$ completion deadline and 1.10 for $6 \times T_{isolate}$ completion deadline), however, significant in cases with loose completion deadlines (1.40 for $9 \times T_{isolate}$ completion deadline and 1.62 for $12 \times T_{isolate}$ completion deadline). The reason is that its overhead for supporting spatial DNN multiplexing is only compensated when there are sufficient DNN multiplexing opportunities. The speedup of HDA over PR increases from 1.39 to 1.50. The increase is significant in cases with urgent completion deadlines but it slows down as the completion deadline becomes looser. This is because HDA only supports multiplexing of up to three DNNs. The speedup of MDA over PR increases from 2.55 to 3.65. The significant speedup results from three aspects: 1) full support of temporal and spatial concurrent processing of DNNs like PL; 2) full support of adaptable dataflow configuration only partially supported by HDA; 3) silicon photonic interconnects that enable low-latency and high-bandwidth communication.

We then fix the predefined completion deadline SLA at $6 \times T_{isolate}$ while varying the DNN arrival rate from $3/M$ to $12/M$. A large arrival rate has similar indications as a large SLA number. We can expect more intense DNN multiplexing, making adherence to completion deadlines challenging while providing more opportunities for better DNN multiplexing setups. The speedup of AI over PR increases insignificantly from 1.13 to 1.23 as AI does not leverage either better DNN multiplexing setups or dataflow flexibility. The speedup of PL over PR increases from 1.07 to 1.53. The increase is trivial in cases with low DNN arrival rates (1.07 for $\lambda = 3/M$ and $\lambda = 6/M$, and 1.09 for $\lambda = 9/M$), however, significant in the case with the highest DNN arrival rate (1.53 for $\lambda = 12/M$). This is because the spatial DNN multiplexing capability of PL is only leveraged when the DNN arrival rate is high. The speedup of HDA over PR increases from 1.19 to 1.50 for its limited spatial DNN multiplexing capability and dataflow flexibility. The speedup of MDA over PR increases notably from 2.31 to 4.71 due to the combined effects of the spatial DNN multiplexing capability, dataflow flexibility, and silicon photonic interconnects, similar to the case of increasing the predefined completion deadline. The speedup comparison results are shown in Fig. 6.

B. Energy Efficiency Results

We define energy efficiency as the reciprocal of the energy consumption of processing a sequence of DNNs. We study the energy efficiency of MDA and other baseline architectures by fixing the DNN arrival rate at $\lambda = 9/M$ while varying the predefined completion deadline SLA from $3 \times T_{isolate}$ to $12 \times T_{isolate}$, and fixing the predefined completion deadline SLA at $6 \times T_{isolate}$ while varying the DNN arrival rate from $3/M$ to $12/M$. The energy efficiency of AI is similar to that of PR as they both only support a fixed dataflow. PL achieves better energy efficiency compared to AI, especially in cases with loose predefined completion deadline (1.68×, $SLA = 12 \times T_{isolate}$ and $\lambda = 9/M$) or high DNN arrival rate (1.60×, $SLA = 6 \times T_{isolate}$ and $\lambda = 12/M$), due to its spatial DNN multiplexing capability, which leads to higher utilization of compute resources. HDA achieves significantly higher energy efficiency than PR, AI, and PL as its sub-accelerators offer three dataflow choices for the multiplexed DNNs. HDA achieves up to 2.8× increase in energy efficiency compared to PR. The proposed MDA architecture achieves up to 7.3 × increase in energy efficiency compared to PR since it provides full flexibility in dataflow selection and extremely energy-efficient communication with silicon photonic interconnects. The energy efficiency comparison results are shown in Fig. 7.

C. SLA Satisfaction Rate Results

We define SLA satisfaction rate as the percentage of the DNNs that adhere to their predefined completion deadlines. Fig. 8 shows that the SLA satisfaction rates for all architectures increase when fixing the DNN arrival rate at $\lambda = 9/M$ and increasing the predefined completion deadline SLA from $3 \times T_{isolate}$ to $12 \times T_{isolate}$, and decrease when fixing the predefined completion deadline SLA at $6 \times T_{isolate}$ and increasing the DNN arrival rate from $3/M$ to $12/M$. In particular, AI and HDA achieve low SLA satisfaction rates as their scheduling algorithms do not prioritize DNNs with high computational demands or short remaining time until their predefined completion deadlines. PR achieves high SLA satisfaction rates as its scheduling algorithm favors DNNs of high priority or short estimated completion time. However, PR still suffers from low SLA satisfaction rates in cases where adhering to predefined

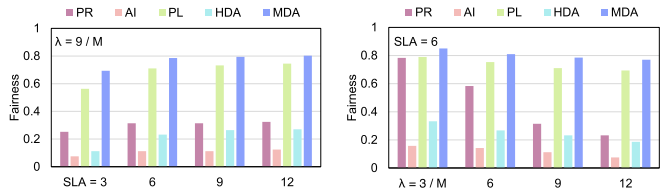


Fig. 9. Fairness comparison when varying SLA (left) and arrival rate λ (right).

completion deadlines is challenging (10%, $SLA = 6 \times T_{isolate}$ and $\lambda = 12/M$). PL achieves the highest SLA satisfaction rate in all the baseline architectures. This is because the assigned compute resources to a DNN in PL are negatively correlated to the estimated remaining time until its predefined completion deadline. When fixing the DNN arrival rate at $\lambda = 9/M$, the proposed MDA architecture achieves a 93% SLA satisfaction rate when assuming a loose predefined completion deadline of $12 \times T_{isolate}$. This number decreases to 73% when assuming the most strict predefined completion deadline of $3 \times T_{isolate}$. Similarly, when fixing the predefined completion deadline at $6 \times T_{isolate}$ and increasing the DNN arrival rate from $3/M$ to $12/M$, the SLA satisfaction rate of MDA decreases from 98% to 81%. The reasons that the SLA satisfaction rate of the proposed MDA architecture can be maintained at a high level and experience an insignificant decrease in cases with intense DNN multiplexing are: 1) DNNs are processed faster on the MDA architecture as demonstrated in Fig. 6; 2) the MDA resource allocation algorithm assigns more compute resources to DNNs with high computational demands or short time left until the predefined completion deadlines.

D. Fairness Results

Fairness [45] is defined as the minimum ratio of relative normalized progress rates of any two concurrently processed DNNs. Fairness is a value that ranges between 0 (no fairness, at least one of the concurrently processed DNNs starves) and 1 (perfect fairness, all concurrently processed DNNs make equal progress). Fig. 9 illustrates that the fairness numbers of MDA and other baseline architectures monotonically increase when fixing the DNN arrival rate at $\lambda = 9/M$ and increasing the predefined completion deadline from $3 \times T_{isolate}$ to $12 \times T_{isolate}$. Furthermore, the fairness numbers monotonically decrease when fixing the predefined completion deadline at $6 \times T_{isolate}$ and increasing the DNN arrival rate from $3/M$ to $12/M$. AI and HDA suffer from low fairness because their scheduling algorithms do not guarantee the equal progress of concurrently processed DNNs. PR especially favors DNNs with short estimated completion time, potentially leading to the starvation of other DNNs. PL and MDA achieve the highest fairness because they both allocate compute resources based on computational demands while favoring DNNs with lagging progress.

VIII. Related Work

Multi-DNN Accelerators: Several prior multi-DNN accelerators [10], [11] support temporal multiplexing of DNNs. They require minimal hardware modifications from original single-DNN accelerators but suffer from low system utilization due to the unmanaged mismatch [12] between hardware resource demand and provision. Other prior multi-DNN accelerators [12], [13] support both temporal and spatial multiplexing of DNNs, however, still suffer from dataflow inflexibility. Some recent multi-DNN accelerators [15], [16] provide a certain level of flexibility in dataflow choices. However, the choices are limited due to the rigidity of metallic-based interconnects. The proposed MDA accelerator architecture exploits silicon photonic interconnects to construct flexible communication fabrics required for multi-DNN processing and free dataflow selection. To our knowledge, this is the first attempt to solve the multi-DNN processing challenges with silicon photonics.

Multi-DNN on GPUs and FPGAs: There is a large body of related work on multi-DNN support on GPU [46], [47], [48] and FPGA [49] platforms. This work focuses on facilitating multi-DNN processing on the hardware accelerator platform where the communication fabrics could play a pivotal role in determining the functionality, performance, and energy efficiency of the platform. That is our motivation to explore the utilization of the disruptive silicon photonics technology.

Dataflow Exploration: Prior work has extensively studied dataflow optimization [4], [6], [27], [50], [51], [52], [53], [54], [55] from temporal data reuse perspective. However, some work targeting single-DNN processing on silicon photonic accelerators [8], [19] has emphasized the necessity of partially shifting the dataflow optimization paradigm from temporal data reuse to spatial data parallelism. This work exhaustively explores the dataflow design space by enumerating the temporal data reuse and spatial data parallelism opportunities, as well as the resulting communication patterns, along all DNN dimensions or combinations of several dimensions.

IX. Conclusion

In this paper, we develop the MDA accelerator architecture which is specially crafted to empower high-performance and energy-efficient concurrent processing of diverse DNNs. MDA includes three notable parts: 1) a resource allocation algorithm that assigns compute resources based on both computational demands and priorities; 2) a dataflow selection algorithm that finds off-chip and on-chip dataflows leading to minimum memory accesses; 3) a flexible silicon photonic network that can be dynamically segmented and adaptively configured to facilitate the communication patterns of concurrently processed DNNs. Simulation studies have shown the effectiveness of the proposed MDA accelerator architecture.

Acknowledgment

The authors would like to thank the anonymous reviewers for the excellent feedback.

References

[1] J. Fowers et al., "A configurable cloud-scale DNN processor for real-time AI," in *Proc. ACM/IEEE Int. Symp. Comput. Archit.*, 2018, pp. 1–14.

[2] U. Gupta et al., "DeepRecSys: A system for optimizing end-to-end at-scale neural recommendation inference," in *Proc. ACM/IEEE Int. Symp. Comput. Archit.*, 2020, pp. 982–995.

[3] F. Romero, Q. Li, N. J. Yadwadkar, and C. Kozyrakis, "INFaaS: A model-less and managed inference serving system," 2019, *arXiv:1905.13348*.

[4] N. P. Jouppi et al., "In-datacenter performance analysis of a tensor processing unit," in *Proc. ACM/IEEE Int. Symp. Comput. Archit.*, 2017, pp. 1–12.

[5] Y. Li and A. Louri, "ALPHA: A learning-enabled high-performance network-on-chip router design for heterogeneous manycore architectures," *IEEE Trans. Sustain. Comput.*, vol. 6, no. 2, pp. 274–288, Second Quarter 2021.

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

[7] Y. Li, A. Louri, and A. Karanth, "Scaling deep-learning inference with chiplet-based architecture and photonic interconnects," in *Proc. ACM/IEEE Des. Automat. Conf.*, 2021, pp. 931–936.

[8] Y. Li, K. Wang, H. Zheng, A. Louri, and A. Karanth, "ASCEND: A scalable and energy-efficient deep neural network accelerator with photonic interconnects," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 69, no. 7, pp. 2730–2741, Jul. 2022.

[9] Y. Li, A. Louri, and A. Karanth, "SPRINT: A high-performance, energy-efficient, and scalable chiplet-based accelerator with photonic interconnects for CNN inference," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 10, pp. 2332–2345, Oct. 2021.

[10] 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. Archit.*, 2020, pp. 220–233.

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

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

[13] J. Lee, J. Choi, J. Kim, J. Lee, and Y. Kim, "Dataflow mirroring: Architectural support for highly efficient fine-grained spatial multitasking on systolic-array NPUs," in *Proc. ACM/IEEE Des. Automat. Conf.*, 2021, pp. 247–252.

[14] H. Kwon, A. Samajdar, and T. Krishna, "MAERI: Enabling flexible dataflow mapping over DNN accelerators via reconfigurable interconnects," in *Proc. ACM Int. Conf. Architectural Support Program. Lang. Operating Syst.*, 2018, pp. 461–475.

[15] 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. Archit.*, 2021, pp. 71–83.

[16] 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. Archit.*, 2022, pp. 814–830.

[17] D. A. B. Miller, "Device requirements for optical interconnects to silicon chips," *Proc. IEEE*, vol. 97, no. 7, pp. 1166–1185, Jul. 2009.

[18] K. Bergman, L. P. Carloni, A. Biberman, J. Chan, and G. Hendry, *Photonic Network-on-Chip Design*. Berlin, Germany: Springer, 2014.

[19] Y. Li, A. Louri, and A. Karanth, "SPACX: Silicon photonics-based scalable chiplet accelerator for DNN inference," in *Proc. IEEE Int. Symp. High-Perform. Comput. Archit.*, 2022, pp. 831–845.

[20] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proc. IEEE*, vol. 105, no. 12, pp. 2295–2329, Dec. 2017.

[21] X. Yang et al., "Interstellar: Using Halide's scheduling language to analyze DNN accelerators," in *Proc. ACM Int. Conf. Architectural Support Program. Lang. Operating Syst.*, 2020, pp. 369–383.

[22] M. Gao, J. Pu, X. Yang, M. Horowitz, and C. Kozyrakis, "TETRIS: Scalable and efficient neural network acceleration with 3D memory," in *Proc. ACM Int. Conf. Architectural Support Program. Lang. Operating Syst.*, 2017, pp. 751–764.

[23] A. H. Atabaki, A. A. Eftekhar, S. Yegnanarayanan, and A. Adibi, "Sub-100-nanosecond thermal reconfiguration of silicon photonic devices," *Opt. Express*, vol. 21, no. 13, pp. 15 706–15 718, Jul. 2013.

[24] M. W. Geis, S. J. Spector, R. C. Williamson, and T. M. Lyszczarz, "Submicrosecond submilliwatt silicon-on-insulator thermooptic switch," *IEEE Photon. Technol. Lett.*, vol. 16, no. 11, pp. 2514–2516, Nov. 2004.

[25] E. Peter, A. Thomas, A. Dhawan, and S. R. Sarangi, "Active microring based tunable optical power splitters," *Opt. Commun.*, vol. 359, pp. 311–315, Jan. 2016.

[26] T. F. de Lima et al., "Machine learning with neuromorphic photonics," *IEEE J. Lightw. Technol.*, vol. 37, no. 5, pp. 1515–1534, Mar. 2019.

[27] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," in *Proc. ACM/IEEE 43rd Int. Symp. Comput. Archit.*, 2016, pp. 367–379.

[28] Y. Li, A. Louri, and A. Karanth, "A silicon photonic multi-DNN accelerator," in *Proc. IEEE/ACM Int. Conf. Parallel Archit. Compilation Techn.*, 2023, pp. 1–12.

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

[30] K. Padmaraju et al., "Intermodulation crosstalk from silicon microring modulators in wavelength-parallel photonic networks-on-chip," in *Proc. Annu. Meeting IEEE Photon. Soc.*, 2010, pp. 562–563.

[31] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi, "CACTI 6.0: A tool to model large caches," *HP Lab.*, vol. 27, pp. 1–24, 2009.

[32] P. Rosenfeld, E. Cooper-Balis, and B. Jacob, "DRAMSim2: A cycle accurate memory system simulator," *IEEE Comput. Archit. Lett.*, vol. 10, no. 1, pp. 16–19, Jan./Jun. 2011.

[33] W. J. Turner et al., "Ground-referenced signaling for intra-chip and short-reach chip-to-chip interconnects," in *Proc. IEEE Custom Integr. Circuits Conf.*, 2018, pp. 1–8.

[34] A. Joshi et al., "Silicon-photonic clos networks for global on-chip communication," in *Proc. IEEE/ACM Int. Symp. Netw.-on-Chip*, 2009, pp. 124–133.

[35] Y. Thonnart et al., "A 10Gb/s Si-photonic transceiver with 150μW 120μs-lock-time digitally supervised analog microring wavelength stabilization for 1Tb/s/mm^2 die-to-die optical networks," in *Proc. IEEE Int. Solid-State Circuits Conf.*, 2018, pp. 350–352.

[36] C. DeCusatis, *Handbook of Fiber Optic Data Communication: A Practical Guide to Optical Networking*. New York, NY, USA: Academic Press, 2013.

[37] A. V. Krishnamoorthy et al., "Computer systems based on silicon photonic interconnects," *Proc. IEEE*, vol. 97, no. 7, pp. 1337–1361, Jul. 2009.

[38] M. Tan and Q. V. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 6105–6114.

[39] V. J. Reddi et al., "MLPerf inference benchmark," in *Proc. ACM/IEEE Int. Symp. Comput. Archit.*, 2020, pp. 446–459.

[40] C. Szegedy et al., "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 1–9.

[41] 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.

[42] G. Huang, Z. Liu, L. V. D. Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 4700–4708.

[43] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. Int. Conf. Learn. Representations*, 2015, pp. 1–14.

[44] J. Redmon and A. Farhadi, "YOLOv3: An incremental improvement," 2018, *arXiv:1804.02767*.

[45] S. Eyerman and L. Eeckhout, "System-level performance metrics for multiprogram workloads," *IEEE Micro*, vol. 28, no. 3, pp. 4–53, May/Jun. 2008.

[46] M. Jeon, S. Venkataraman, A. Phanishayee, J. Qian, W. Xiao, and F. Yang, "Analysis of large-scale multi-tenant GPU clusters for DNN training workloads," in *Proc. USENIX Annu. Tech. Conf.*, 2019, pp. 947–960.

[47] F. Yu et al., "Automated runtime-aware scheduling for multi-tenant DNN inference on GPU," in *Proc. IEEE/ACM Int. Conf. Comput. Aided Des.*, 2021, pp. 1–9.

[48] J. Mohan, A. Phanishayee, J. Kulkarni, and V. Chidambaram, "Looking beyond GPUs for DNN scheduling on multi-tenant clusters," in *Proc. USENIX Symp. Operating Syst. Des. Implementation*, 2022, pp. 579–596.

[49] S. Zeng et al., "Serving multi-DNN workloads on FPGAs: A coordinated architecture, scheduling, and mapping perspective," *IEEE Trans. Comput.*, vol. 72, no. 5, pp. 1314–1328, May 2023.

[50] Z. Du et al., "ShiDianNao: Shifting vision processing closer to the sensor," in *Proc. ACM/IEEE Int. Symp. Comput. Archit.*, 2015, pp. 92–104.

[51] S. Chakradhar, M. Sankaradas, V. Jakkula, and S. Cadambi, "A dynamically configurable coprocessor for convolutional neural networks," in *Proc. ACM/IEEE Int. Symp. Comput. Archit.*, 2010, pp. 247–257.

[52] L. Cavigelli, D. Gschwend, C. Mayer, S. Willi, B. Muheim, and L. Benini, "Origami: A Convolutional Network Accelerator," in *Proc. ACM Great Lakes Symp. VLSI*, 2015, pp. 199–204.
[53] T. Chen et al., "DianNao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," in *Proc. ACM Int. Conf. Architectural Support Program. Lang. Operating Syst.*, 2014, pp. 269–284.
[54] 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. Architectural Support Program. Lang. Operating Syst.*, 2019, pp. 807–820.
[55] 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.

Yuan Li (Member, IEEE) received the BS degree in physics from the University of Science and Technology of China in 2010, the MS degree in microelectronics from the University of Newcastle upon Tyne in 2011, and the PhD degree in computer engineering from the George Washington University, in 2022. His research interests include AI hardware acceleration, chiplet-based heterogeneous integration, and on-chip interconnection network.

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, which he joined in 2015. He is also the director of the 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, 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 scalable 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 the 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 manycore architectures." He is currently the editor-in-chief of *IEEE Transactions on Computers*.

Avinash Karanth (Senior Member, IEEE) received the BE degree in electronics and communications from the Manipal Institute of Technology, Mangalore University in Feb. 2000, and the MS and PhD degrees from the Electrical and Computer Engineering Department, The University of Arizona, in 2003 and 2006, respectively. Presently, he is the Joseph Jachinowski professor with the School of Electrical Engineering and Computer Science at Ohio University in Athens, Ohio. He directs the Technologies for Emerging Computer Architecture Lab (TEAL), Ohio University. His research interests include computer architecture, optical interconnects, Network-on-Chips (NoCs) and emerging technologies such as nanophotonics, 3D, and wireless interconnects. He is the recipient of the NSF CAREER Award in 2011, the Presidential Research Scholar Award in 2017, the Best Paper Award at the ICCD 2013 conference, and his papers have been nominated for Best Paper at the IEEE Symposium on Network-on-Chips (NoCs) in May 2010 and the IEEE Asia & South Pacific Design Automation Conference (ASP-DAC) in January 2009. He is a member of ACM.