Heterogeneous Dataflow Accelerators for Multi-DNN Workloads

Hyoukjun Kwon*†, Liangzhen Lai†, Michael Pellauer‡, Tushar Krishna*, Yu-Hsin Chen†, Vikas Chandra†
*Georgia Insitute of Technology, †Facebook, ‡NVIDIA
*hyoukjun@gatech.edu, tushar@ece.gatech.edu ,
†{hyoukjunkwon, liangzhen, yhchen, vchandra}@fb.com, ‡mpellauer@nvidia.com

***Abstract*—Emerging AI-enabled applications such as augmented and virtual reality (AR/VR) leverage multiple deep neural network (DNN) models for various sub-tasks such as object detection, image segmentation, eye-tracking, speech recognition, and so on. Because of the diversity of the sub-tasks, the layers within and across the DNN models are highly heterogeneous in operation and shape. Diverse layer operations and shapes are major challenges for a fixed dataflow accelerator (FDA) that employs a fixed dataflow strategy on a single DNN accelerator substrate since each layer prefers different dataflows (computation order and parallelization) and tile sizes. Reconfigurable DNN accelerators (RDAs) have been proposed to adapt their dataflows to diverse layers to address the challenge. However, the dataflow flexibility in RDAs is enabled at the cost of expensive hardware structures (switches, interconnects, controller, etc.) and requires per-layer reconfiguration, which introduces considerable energy costs.**

Alternatively, this work proposes a new class of accelerators, heterogeneous dataflow accelerators (HDAs), which deploy multiple accelerator substrates (i.e., sub-accelerators), each supporting a different dataflow. HDAs enable coarser-grained dataflow flexibility than RDAs with higher energy efficiency and lower area cost comparable to FDAs. To exploit such benefits, hardware resource partitioning across sub-accelerators and layer execution schedule need to be carefully optimized. Therefore, we also present Herald, a framework for co-optimizing hardware partitioning and layer scheduling. Using Herald on a suite of AR/VR and MLPerf workloads, we identify a promising HDA architecture, Maelstrom, which demonstrates 65.3% lower latency and 5.0% lower energy compared to the best fixed dataflow accelerators and 22.0% lower energy at the cost of 20.7% higher latency compared to a state-of-the-art reconfigurable DNN accelerator (RDA). The results suggest that HDA is an alternative class of Pareto-optimal accelerators to RDA with strength in energy, which can be a better choice than RDAs depending on the use cases.

I. Introduction

The success of deep learning over the past few years has led to the development of breakthrough applications such as augmented and virtual reality (AR/VR) [1] and autonomous driving [2], [3], [4]. These applications employ not one, but multiple deep neural networks (DNN) internally for various tasks to collaboratively achieve state-of-the-art operational performance [1] of the application. For example, a VR application includes sub-tasks such as object detection to prevent users from conflicting with nearby obstacles, hand tracking, and hand pose estimation for user inputs, eye-tracking for foveated rendering, and so on [5], [6], [7], [8].

[1]Since performance is an overloaded term, we distinguish computational (i.e., latency, throughput, etc.) and operational (i.e., the accuracy of DNN classification) performance in this paper.

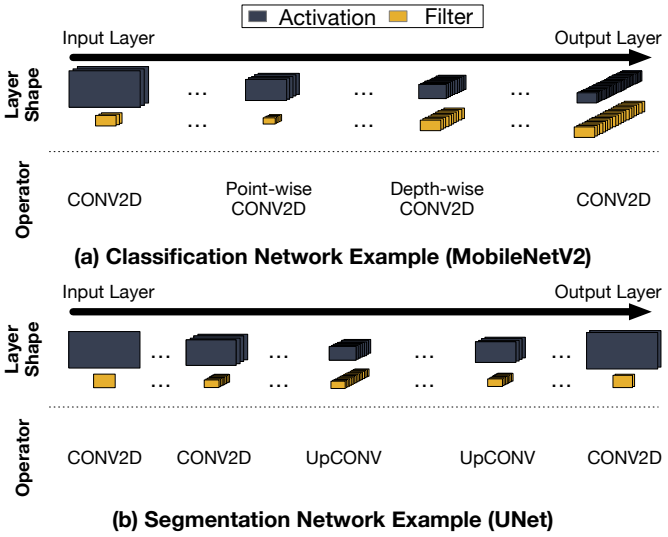

Fig. 1. A high level overview of layer shape (i.e., tensor shapes) of (a) classification networks such as Resnet50 [9] and MobileNetv2 [10] and (b) segmentation networks such as UNet [11].

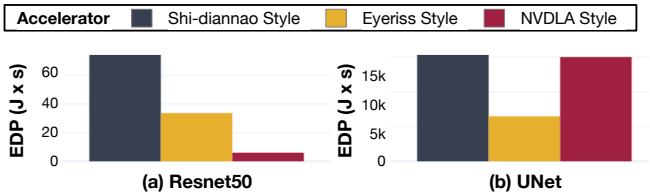

Fig. 2. EDP estimation of DNN accelerators with output-stationary (ShiDianNao) [12], weight-stationary (NVDLA) [13], and row-stationary (Eyeriss) [14] style dataflows for running Resnet50 and UNet. For a fair comparison, we choose 256 PEs and 32GBps NoC bandwidth for all accelerators and model them within a common framework MAESTRO [15] that estimates the energy and runtime based on data reuse facilitated by the dataflow.

We list some of the DNNs used in AR/VR applications in Table I. Despite all these DNNs being Convolutional Neural Network (CNN)-based, they exhibit high heterogeneity - both in layer shapes and operations, depending on the task. E.g., across the layers in the example DNN models, the largest channel-activation size ratio, which is an indicator of layer shapes, is 315076× larger than the smallest one. In terms of operations, in addition to CONV2D, these DNNs rely on operators

TABLE I

HETEROGENEITY IN DNN MODELS USED IN AR/VR WORKLOADS [1]. FOR WORKS WITHOUT MODEL NAMES, WE NAME THEM TO REFER TO THOSE WORKS IN THE REST OF THE PAPER. THE CHANNEL-ACTIVATION SIZE RATIO IS AN ABSTRACTION OF LAYER SHAPE, WHICH REFERS TO THE NUMBER OF CHANNELS DIVIDED BY ACTIVATION SIZE (WIDTH OR HEIGHT). CONV2D, PWCONV, DWCONV, SKIP-CON, UPCONV, AND CONCAT REFER TO 2D CONVOLUTION, POINT-WISE 2D CONVOLUTION, DEPTH-WISE CONVOLUTION, SKIP CONNECTION, UP-SCALE CONVOLUTION, AND CONCATENATION, RESPECTIVELY.

Task	Model	Channel-Activation Size Ratio	Layer Operations
Object Detection	MobileNetV2 [10]	Min: 0,013, Median: 13.714, Max: 1280	CONV2D, PWCONV, DWCONV, Skip-Con.
Object Classification	Resnet50 [9]	Min: 0.013, Median: 18.286, Max: 292.571	CONV2D, FC, Skip-Con.
Hand Tracking	UNet [11]	Min: 0.002, Median: 1.855, Max: 34.133	CONV2D, FC, UPCONV, Concat.
Hand Pose Estimation	Br-Q HandposeNet [16]	Min: 0.016, Median: 1024, Max: 1024	CONV2D, FC
Depth Estimation	Focal Length DepthNet [17]	Min: 0.013, Median: 4.571, Max: 4096	CONV2D, FC, UPCONV

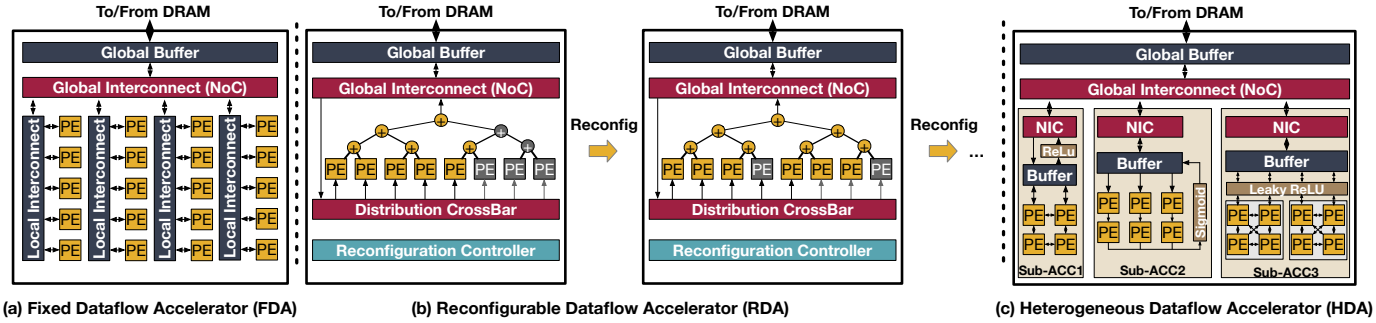

Fig. 3. Examples of fixed, reconfigurable, and heterogeneous dataflow accelerators.

such as depth-wise, transpose, and up-scale convolutions. The heterogeneity is shown qualitatively in Figure 1 and explained in Subsection II-A.

Such extreme layer heterogeneity introduces major efficiency (i.e., latency and energy efficiency) challenges to DNN accelerators since they are often over-specialized for a specific set of DNN layers, which provides them an efficiency boost in the first place. This over-specialization is based on the *dataflow* choice in the accelerator. We can quantitatively observe the impact of dataflow choices in Figure 2, which compares energy-delay-product (EDP) of Shi-diannao [12], NVDLA [13], and Eyeriss [14] style accelerators across two different DNN models. NVDLA's dataflow exploits parallelism across input and output channels, which enables to achieve near roof-line throughput and thus low EDP for CONV2D layers with deep channels (such as those in ResNet50), as shown in Figure 2 (a). However, when running CONV2D layers with a small number of channels dominant in UNet, NVDLA suffers from compute unit under-utilization, which leads to low throughput and high energy, as results in Figure 2(b) show. In contrast, a row-stationary style dataflow like Eyeriss parallelizes the computation over activation rows enables high PE utilization on such CONV2D layers. In other words, tuning an accelerator's dataflow for specific layers can lead to inefficiency across other layers. We call these existing approaches *Fixed Dataflow Accelerators* (FDAs).

The observation above is not new. Multiple prior papers [15], [18], [19], [20] have pointed to the fact that the optimal dataflow and the tile size (together called a *mapping*) choices are highly dependent on the layer shape and operation, and one dataflow/mapping choice is not ideal for all layers of a model. Flexible accelerators, i.e., accelerators that support multiple dataflows, have been studied in the past for this challenge. They include coarse-grained reconfigurable architecture (CGRA) style ASIC accelerators [18], [21], [22]. We term such approaches *reconfigurable dataflow accelerators* (RDAs). A key challenge with RDAs is that the flexibility is enabled at the cost of extra hardware components (switches and wires) that are a cause for concern for deployment under stringent energy constraints in edge, mobile, and cloud devices (e.g., MAERI [18] required 11.7% more energy, on average, compared to NVDLA-style FDA in our evaluation). Moreover, reconfiguring for the optimal mapping for each layer [23] would also add additional latency and power costs at the end of each layer.

In this work, we propose a new class of DNN accelerators called *heterogeneous dataflow accelerators* (HDAs). HDAs provide flexibility by employing multiple sub-accelerators, each tuned for a different dataflow, within an accelerator chip. HDAs provide two important features: (i) *dataflow flexibility*, enabled by scheduling each layer from the multiple DNN models on the most efficient sub-accelerator for each layer, as long as possible. (ii) *high utilization*, enabled by scheduling multiple layers from different models across the sub-accelerators simultaneously.

Using four example HDA architectures in the evaluation, we demonstrate that the HDA approach offers a promising mechanism for enabling dataflow flexibility similar to RDAs while staying within the area-power budget of FDAs and being more robust to workload changes. In our evaluation, HDAs with the best EDP for each experiment provided 73.6% lower energy-delay product (65.3% latency and 5.0% energy benefits) across evaluated multi-DNN workloads, compared to the best monolithic designs we evaluate.

We summarize the contribution of this paper as follows:

- This is the first work to propose the concept of HDAs for DNN acceleration.

- We propose a hardware and schedule co-design space exploration (DSE) algorithm that searches for (i) optimized hardware resource distribution across sub-accelerators and (ii) optimized layer execution schedules on the sub-accelerators for a given multi-DNN workload
- We codify the DSE algorithm and implement Herald, which can be used as by architects at **design time** by running (i) and (ii) together, or by compilers as a scheduler by running (ii) at **compile time**.
- We identify a novel HDA partitioning strategy that employs NVDLA [13] and Shi-diannao [12] style dataflows for unique benefits. We name this accelerator architecture **Maelstrom** and explore the scalability over edge, mobile, and cloud scenarios. On average, across three multi-DNN workloads and three scalability scenarios, Maelstrom demonstrates 65.3% lower latency and 5.0% lower energy compared to the best fixed dataflow accelerators, 63.1% lower latency and 4.1% lower energy compared to the homogeneous multi-DNN [24]-style accelerators, and 20.7% higher latency and 22.0% lower energy compared to an exiting reconfigurable accelerator [18].

II. Background and Motivation

A. Heterogeneous Multi-DNN Workloads

To achieve high-quality (classification, prediction, etc.) results, many applications now employ DNNs as their backbone to perform tasks like face recognition [25], image segmentation [11], [26], depth estimation [17], and so on. Combining DNNs for such tasks, emerging applications such as AR/VR implement complex functionalities, which lead to multi-DNN workloads [1]. These sub-task DNNs are significantly diverse, as shown in Table I. The diversity of models naturally leads to high variations in layer (1) shape and (2) operations, which constructs heterogeneous multi-DNN workloads.

1) Layer Shape: Classification networks such as Resnet [9] or MobileNetV2 [10] gradually reduce the resolution of activation because their goal is to extract a classification vector where each entry represents the probability of each class. Also, classification networks increase the number of channels to exploit as many features as possible for accurate classification. Therefore, layers in classification networks have high-resolution activation and shallow channels in early layers and low-resolution activation and deep channels in late layers, as illustrated in Figure 1 (a).

In contrast, segmentation networks such as UNet [11] need to restore the original resolution of activation because their goal is to generate masks over target objects in the input image. However, segmentation networks still need to extract as many features as those in classification networks for high accuracy. Therefore, segmentation networks first follow the same trend as classification networks until the mid-layer. Afterward, segmentation networks reduce the number of channels and gradually restore the resolution of activation using up-scaling operators such as up-scale convolution or transposed convolution. As a result, layer shapes in segmentation networks follow the trend illustrated in Figure 1 (b).

```
for(k1=0; k1<K1; k1++)
 pfor(k0=0; k0<K0; k0++)
  for(c1=0; c1<C1; c1++)
   for(y1=0; y1<Y1; y1++)
    for(x1=0; x1<X1; x1++)
     pfor(c0=0; c0<C0; c0++)
      for(r1=0; r1<R; r1++)
       for(s1=0; s1<S; s1++)
        for(y0=0; y0<Y0; y0++)
         for(x0=0; x0<X0; x0++)
          for(r=0; r0<1; r++)
           for(s=0; s0<1; s++) {
            k=k1*K0 + k0; c=c1*C0 + c0;
            … x = x1*X0 + x0;
            Output[k][y][x] +=
            Input[c][y+r][x+s] * Filter[k][c][r][s]; }
```

(a) NVDLA Style Dataflow

```
for(k1=0; k1<K1; k1++)
 for(k0=0; k0<K0; k0++)
  for(c1=0; c1<C1; c1++)
   for(y1=0; y1<Y1; y1++)
    for(x1=0; x1<X1; x1++)
     for(c0=0; c0<C0; c0++)
      pfor(y0=0; y0<Y0; y0++)
       pfor(x0=0; x0<X0; x0++)
        for(r=0; r<R; r++)
         for(s=0; s<S; s++) {
          k=k1*K0 + k0; c=c1*C0 + c0;
          … x = x1*X0 + x0;
          Output[k][y][x] +=
          Input[c][y+r][x+s] * Filter[k][c][r][s]; }
```

(b) Shi-diannao Style Dataflow

Fig. 4. Loop nest representation of dataflows from recent accelerators [12], [14]. K and C refer to output and input channels, Y and X refer to input row and column, and R and S refer to filter row and column, respectively. Numbers after loop variables indicate tile levels, and pfor refers to a parallel for loop. We omit edge case handling for simplicity.

2) Layer Operation: As listed in the layer operation column of Table I, layer operations in heterogeneous multi-DNN workloads are diverse. For example, MobileNetV2 performs depth-wise separable convolution [10], which consists of two point-wise convolutions and a depth-wise convolution.

Based on the shape and operation, each layer prefers different dataflow styles and hardware [15], which makes such workloads challenging for fixed dataflow accelerators (FDAs). We clarify the definition of dataflow and mapping and discuss why each layer prefers different dataflows next.

B. Dataflow and Mapping

We collectively refer to loop ordering and spatial unrolling (or partitioning) as dataflow [14], [20]. Dataflows are often represented in a loop-nest form [27], as shown in Figure 4, loop nest with loop bounds are unfilled. From a base loop nest without any loop transformation, a series of loop interchange and parallelization modifies how we compute DNN operations while preserving what we compute. The loop nests in Figure 4 are results of such loop transformations. By providing valid loop bounds to the representation (i.e., loop blocking factor), we obtain "mapping," which indicates an instance of dataflow, which contains full information to map a DNN operation on an accelerator [28]. In DNN accelerators, the mapping dictates the latency and energy consumption because it determines the number of buffer accesses, degree of parallelization (mapping utilization of PEs), buffer size requirements, and so on [14], [15], [19], [20], [21].

When constructing a mapping from a dataflow, the set of valid loop bounds (loop blocking factors) are constrained by the sizes of each layer dimension (i.e., out-of-bound) and layer operation (e.g., depth-wise convolution does not accumulate partial sums across input channels unlike CONV2D). Such constraints to loop bounds from layer shape and operation determine a set of available mappings from each dataflow, which appears as the preferences to layers. To further understand such an aspect, we use two example FDAs with NVDLA and Shi-diannao

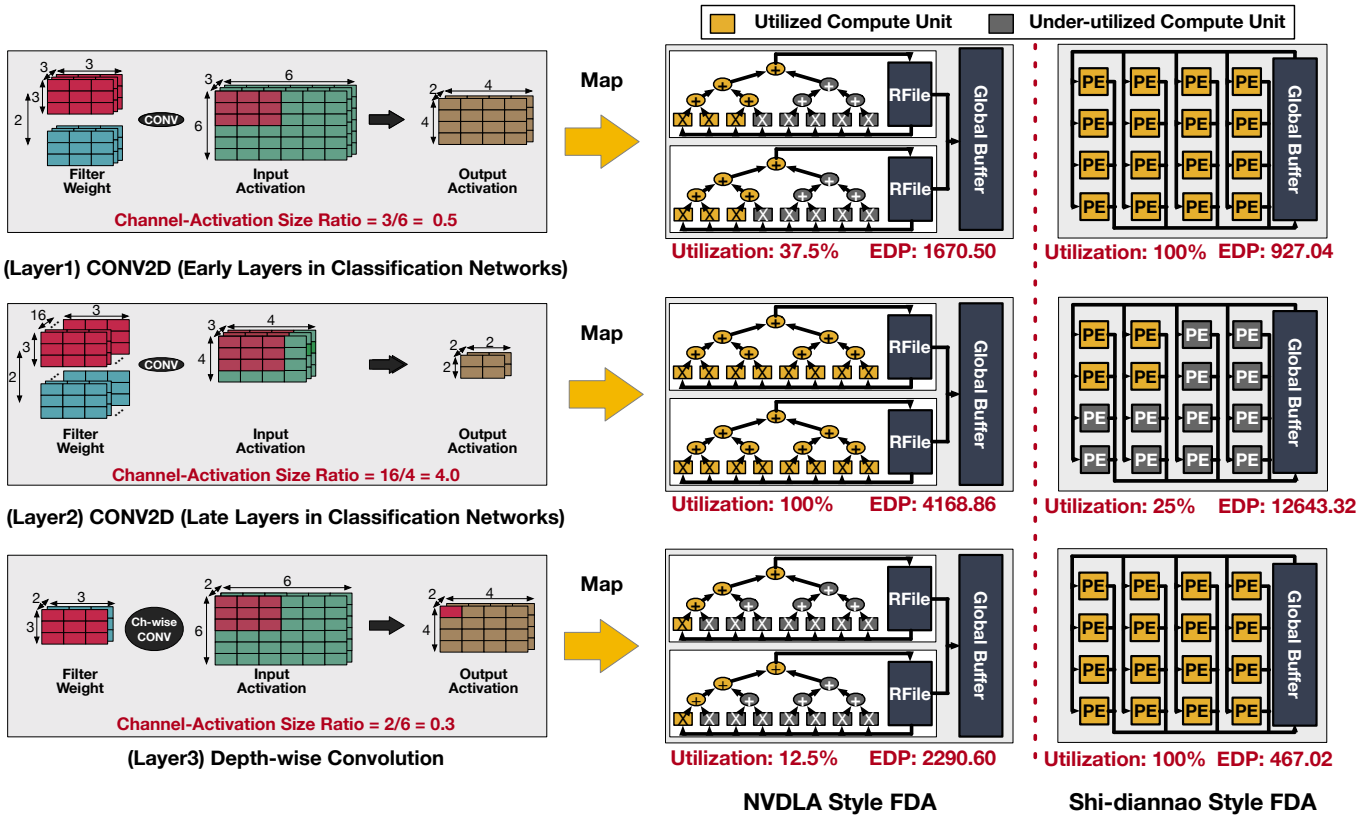

Fig. 5. The impact of dataflow styles on efficiency. We show three example layer execution scenarios on NVDLA and Shi-diannao style FDAs. The utilization refers to the mapping utilization of compute units, which refers to the ratio of the number of PEs a mapping utilizes and the total number of PEs in an accelerator.

dataflow styles and three example layers shown in Figure 5. For simplicity, we select the minimum loop blocking factor to construct mappings for each dataflow.

Those two example accelerators have distinct approaches to compute MAC operations in DNNs. As illustrated in Figure 5, a Shi-diannao style accelerator parallelizes the computation over output activation row and column dimensions using an output-stationary style dataflow, which exploits output and convolutional reuse. Unlike the example Shi-dianao style accelerator, the example NVDLA style accelerator in Figure 5 parallelizes the computation over input and output channels using a weight-stationary style dataflow, which exploits filter weight reuse. Such differences in parallelization strategies of the example dataflows result in dramatically different mapping utilization of compute units, as shown in Figure 5. We use three example layers presented in Figure 5 to show the impact of mappings. Layer 1 and 2 are CONV2D operations with the aspect ratio of early and late layers in classification network introduced in Figure 1 (a), respectively. Layer 3 is a depth-wise CONV2D operation with the same layer size as Layer 1.

Based on the parallelization strategies of each example accelerator and layer sizes, we can observe dramatically different PE utilization as shown in Figure 5. We use MAESTRO [15] cost model for DNN accelerators to estimate the latency and energy and compute energy-delay product (EDP) as one of the indicators of overall efficiency, as shown in Figure 5. In the combination of the differences in utilization and data reuse strategies, two example accelerators result in dramatically different EDPs, which implies distinct preferences of two example accelerators to the layers. In addition to the mapping utilization, each of the example mappings has dramatically different memory/network-on-chip(NoC) bandwidth requirements, buffer size requirements, and so on, which also varies based on the layer shape and operations in a different degree [15].

Therefore, no single dataflow style is good for all the layers, and we need to optimize the dataflow for each layer in target workloads to maximize the efficiency of an accelerator. However, when the target workload is heterogeneous, the common practice that optimizes the dataflow for the average case of the workload can result in a consistently inefficient mapping for all the layers in the workload, which is one of the major challenges for DNN acceleration for emerging applications with multiple DNN models. We discuss available accelerator options to deal with such a problem with a general introduction to DNN accelerators next.

III. Heterogeneous Dataflow Accelerators (HDAs)

We propose heterogeneous dataflow accelerators (HDAs) that deploy multiple FDA instances within a chip, each running a different dataflow, as illustrated in Figure 3 (c). This approach eliminates extra hardware costs for reconfigurability and enables dataflow flexibility by enabling the selection of

appropriate sub-accelerators for each layer. To maximize the efficiency of computation, HDAs by default assign layers to a sub-accelerator with the most preferred dataflow style for each layer. However, to exploit more benefits by HDAs under the efficiency drop from smaller sub-accelerators than full FDAs or RDAs, HDAs require new design considerations that did not exist in FDAs or RDAs. We first discuss such design considerations and formally define the HDA architecture based on them. Then, we highlight aspects of HDAs that provide benefits and challenges.

A. Design Considerations and Definition of HDA

To design an HDA, we need to (1) select dataflows for sub-accelerators, (2) determine how to partition existing hardware resources across sub-accelerators, and (3) find a legal layer execution schedule that satisfies layer dependence and memory size constraints. We discuss those three design considerations.

Dataflow Selection for Sub-accelerators. As we show in Section V, the dataflow styles to build sub-accelerators of an HDA is crucial for overall efficiency. To maximize the benefits from dataflow flexibility, the dataflow styles of sub-accelerators need to be sufficiently different so that the resulting HDA can adapt to different layers with diverse shapes and operations.

Hardware Resource Partitioning. As a PE partitioning example in Figure 6 shows, evenly distributed hardware resources across sub-accelerators often lead to sub-optimal HDA design points. This shows that resource partitioning is a non-trivial optimization problem. Also, the optimal distribution depends on workloads and selected dataflows, which makes determining hardware resource distribution further challenging.

Layer Scheduling. Because HDAs include multiple accelerator instances, we need to determine the layer execution schedule across sub-accelerators, which is important for exploiting layer parallelism. A scheduler must check if generated schedules are valid in terms of layer dependence and memory constraints. In addition to that, the scheduler needs to optimize overall latency and energy, not those of each layer (i.e., global optimization for the entire HDA, not local optimization for each sub-accelerator). Designing a scheduler satisfy all of the aforementioned requirements is challenging, and boosting the scheduler's speed to facilitate fast hardware and schedule co-design space exploration (DSE) of HDAs is another challenge. Based on the design considerations, we formally define the HDA architecture as follows:

Definition 1. HDA Architecture
For given N_d dataflow styles, D = $\{\delta_1, \delta_2, ..., \delta_{N_d}\}$, total number of PEs, N_{PE}, total global NoC bandwidth BW_G, an HDA architecture H is defined as follows:
$H = \{(\delta_i, N_i, BW_i) \mid 1 \leq i \leq n \wedge \sum N_i = N_d \wedge \sum BW_i = BW_G\}$

The definition specifies the dataflow styles for each sub-accelerator, PE and bandwidth partitioning across sub-accelerators, and the total number of sub-accelerators, which fully specifies all the HDA-specific design parameters.

B. Benefits of HDAs

When optimized properly, HDAs provide latency and energy benefits based on the following three aspects.

Selective Scheduling. Because each layer prefers different dataflow and hardware, running each layer on its most preferred sub-accelerator in an HDA is an effective solution to maximize overall efficiency.

Layer Parallelism. Unlike most FDAs and RDAs that run one layer and another, HDAs can simultaneously run multiple layers of different models. By this approach, an HDA can overlap the latency of multiple models, which leads to latency hiding among DNN models reducing overall latency.

Low Hardware Cost for Dataflow Flexibility. Because HDA employs FDA style sub-accelerators, HDAs do not involve the costs for reconfigurability like RDAs.

C. Challenges for HDAs

As we discussed in Subsection III-A, optimizing HDA requires various considerations at once, which makes the HDA design challenging. We discuss three aspects of the challenges.

Reduced Parallelism for Each Layer. Given the same number of PEs for an FDA and an HDA, sub-accelerators in the HDA have smaller numbers of PEs than the FDA since hardware resources need to be distributed (or partitioned) across sub-accelerators. Therefore, the maximum degree of parallelism for each sub-accelerator decreases compared to an FDA or an RDA with the same number of PEs in total. A smaller number of PEs and less available parallelism can lead to not only higher latency but also higher energy consumption since the amount of spatial data reuse (i.e., multicast factor) also decreases [15].

Shared Memory and NoC Bandwidth. Because multiple sub-accelerators share a global scratchpad memory and global NoC, those resources either need to be time-multiplexed or hard-partitioned across sub-accelerators. Like the smaller number of PEs in sub-accelerators of an HDA compared to FDAs or RDAs can lead to potential inefficiencies, lower memory and NoC bandwidth can also lead to higher latency. To mitigate this, exploiting as much layer parallelism across sub-accelerators as possible is a key, which motivates a good scheduler.

Scheduling under Memory and Dependence Constraints to Minimize Dark Silicon. One of the important aspects of HDAs to enhance overall efficiency is layer parallelism to exploit as many compute units in sub-accelerators as possible, which requires a good layer scheduler. In addition to maximizing the utilization, the layer scheduler needs to consider constraints from layer dependence and global memory size. The scheduler needs to assign layers on the most preferred sub-accelerator to exploit the benefits of flexible dataflow. However, such a simple greedy method that seeks a locally optimal schedule can result in a globally sub-optimal schedule, which is another challenge for the scheduler.

To enable more benefits over discussed challenges, HDA needs to be carefully optimized. Because many design considerations need to be considered at once, and determining one affects the optimal combinations of others, we need a systematic approach to optimize HDAs. Therefore, we develop a hardware

and schedule co-design space exploration (DSE) algorithm for HDAs that co-optimize all the design considerations in hardware and schedule. We codify the algorithm and implement Herald, an HDA optimization framework, which automates HDA design tailored for user-specified target models and outputs estimated latency and energy using the co-optimized design. We discuss the DSE algorithm and its implementation, Herald, next.

IV. Design Space Exploration Algorithm for HDAs

We discuss the HDA co-design space exploration (DSE) algorithm that co-optimizes hardware resource partitioning across sub-accelerators and layer execution schedule. We first discuss the execution model of HDA and the latency/energy estimation methodology we use. Then we discuss the implementation of the DSE algorithm, Herald.

A. Execution Model

We target layer granularity execution on each sub-accelerator of HDAs to (1) exploit significantly different dataflow preference of layers we discussed in Subsection II-B and (2) more fine-grained scheduling (e.g., parallelize computation tiles of one layer across multiple sub-accelerators) results in high control, synchronization, and scheduling overhead. We assume the following execution steps of accelerators in Herald.

1) Fetch global buffer level filter weight tile from DRAM to a global buffer.
2) Distribute sub-accelerator level filter weight tiles to sub-accelerators based layer execution schedule.
3) Fetch global buffer level activation tile from DRAM to the global buffer.
4) Stream sub-accelerator level activation tiles into their corresponding sub-accelerators based on layer execution schedule.
5) Store streamed-out output activation from each sub-accelerator to the global buffer.
6) Overlapping the computation and data fetch from DRAM, pre-fetch next activation and filter tiles (double buffering). during sub-accelerators compute output activation, fetch next filter values from DRAM and send the filter values to the next accelerator (assumes double-buffering).
7) When a sub-accelerator finishes executing a layer, stream output activation stored in the global buffer as input activation of the next layer.
8) Repeat above processes until processing all the layers of all the models.

For steps 3 and 6, activation is stored in DRAM and loaded in a tiled manner specified by the mapping in the target sub-accelerator if the buffer size is not sufficient to store the entire activation. When output activation is committed to the global buffer, Herald by default assumes a rearrange buffer that adjusts the data layout for the next layer if it runs on another sub-accelerator with a different dataflow style. In the evaluation, we select dataflows that have the same inner-loop order so that we can maintain the same data layout, which eliminates sub-accelerator context change overheads from different data layouts. For the data layout and miscellaneous context change overheads, Herald also provides an option to specify the latency and energy penalties for them.

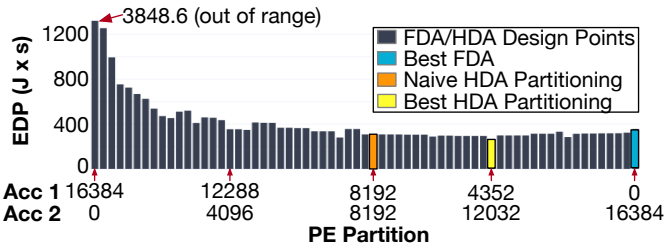

Fig. 6. The impact of PE partitioning upon a cloud accelerator listed in Table IV with two sub-accelerators (ACC1: Shi-diannao style, ACC2: NVDLA style) with naive bandwidth partitioning. We use AR/VR-A workload presented in Section V. The left- and right-most represents ACC1 and ACC2 FDA designs.

B. Latency and Energy Estimation

To guide the design space exploration, we need a cost model that estimates the latency and energy for a given HDA, a workload, and a schedule. We use MAESTRO as a base cost model, which is a validated cost model for monolithic DNN accelerators (i.e., FDAs and RDAs) with any dataflow, which reported 96.1% accuracy against RTL simulation [18] and real processing time measured on a chip [27]. We extend MAESTRO [15], [28] cost model to support multi-DNN sub-accelerator environments, including HDAs.

The implementation of the DSE algorithm, Herald, models the memory requirement for the global buffer and data movement from/to the global buffer to/from sub-accelerator buffers. The modeling method follows the same methodology proposed by MAESTRO, which identifies the amount of reuse and computing activity counts based on them (for energy) and communication/computation delay considering reuse (for latency). In addition to the same analytic equations, Herald considers the layer execution schedule generated by our scheduler discussed in Subsection IV-D by modeling non-synchronized execution of sub-accelerators (i.e., each sub-accelerator start processing a layer as soon as input data are available). For estimating the latency and energy of each sub-accelerator run, we exploit the original MAESTRO cost model.

Next, we discuss two major steps in our DSE algorithm for HDAs: hardware resource partitioning and layer scheduling.

C. Hardware Resource Partitioning Optimization

Unlike FDAs and RDAs fully exploit hardware resources and implement a monolithic accelerator substrate, HDAs need to distribute such resources across sub-accelerators. However, evenly distributing those resources does not yield the most optimal HDAs because each sub-accelerator's dataflow style has different bandwidth requirements [29] and efficiency for a given number of PEs [15].

To quantitatively show the non-trivial design space of hardware resource partitioning, we show an example in Subsection IV-D that shows the impact of PE partitioning over two

sub-accelerators in a 16K-PE-HDA. We show the pure impact of PE partitioning using naive bandwidth partitioning (128/128 GBps) with layer execution schedules generated by Herald's scheduler we discuss in Subsection IV-D. We observe that evenly partitioned PEs (8K/8K) result in a sub-optimal design point, with 17% higher EDP than the optimal PE partition. Therefore, we explore the PE partitioning space to identify the optimal partitioning strategy. In addition to PEs, we also explore the global memory/NoC bandwidth partitioning, which models the hard-partitioning of global NoC wires dedicating partitioned wires for each sub-accelerator. Such an approach is to (1) provide the right amount of bandwidth for each sub-accelerator based on the fact that the bandwidth requirement depends on dataflow choices [15], [29] and (2) minimize hardware costs for fully flexible (i.e., all-to-all) global NoC.

To explore the partitioning choices for given dataflow styles, we implement an algorithm that explores HDA architectures defined in Definition 1. The DSE algorithm, by default, performs an exhaustive search based on user-specified search granularity. However, the DSE algorithm also supports binary sampling or random search, which significantly reduces the search time at the cost of possible loss of globally optimal design points.

D. Layer Execution Schedule Optimization

The main challenge for a scheduling algorithm for HDA is the massive space of schedules. For example, 2.54×10^{21} possible layer execution schedules exist for AR/VR-A workload in Table II even if we only consider permutation of the layers on a single accelerator. To deal with such a large search space, we develop a set of heuristics that exploit the characteristics of DNN workloads to reduce the scheduling overhead. Combining the heuristics, we implement our scheduling algorithm presented in Figure 7. We discuss the heuristics we employ.

Dataflow preference-based layer assignment on sub-accelerators. Exploiting the preferences toward dataflows, our scheduler, by default, assigns each layer on a sub-accelerator with the most-preferred dataflow. Our scheduler implements such greedy methods, and users can select the metric (e.g., EDP, energy, latency, and so on) for them.

Because greedy methods often result in a locally optimal schedule (i.e., optimal for each layer) and miss globally optimal one, our scheduler implements a feedback loop for global load-balancing from the initial schedule constructed after layer ordering. When the scheduler detects an unbalanced load across sub-accelerators, the scheduler explores alternative layer assignment that reduces overall costs (EDP, energy, latency, and so on, specified by users). Users can specify the maximum allowed load-unbalancing factor, the largest latency across sub-accelerators divided by the smallest one. Our scheduler detects an unbalanced load based on the factor.

Heuristic-based Initial Layer Ordering. Our scheduler exploits the characteristics of layer dependence of multi-DNN workloads: layers have mostly (i) linear dependence chain within each model, and (ii) layers are independent across models. Exploiting (i), our scheduler implements a depth-first layer ordering algorithm, which schedules all the layers in a DNN model first and moves on to another. Exploiting (ii), our scheduler implements a breadth-first layer ordering algorithm, which interleaves the layer execution of each DNN model. Those two layer ordering algorithms do not provide the optimal layer order, but they enable to quickly construct a valid initial layer execution order. We describe the layer assignment and ordering algorithm in Figure 8.

Eliminating Redundant Idle Time In Initial Schedules via Post-processing. The initial schedule based on simple depth-first or breadth-first layer ordering often has unnecessary idle time based on bad layer execution order. The post-processing algorithm fixes such inefficiencies with O(mn) complexity (m: total number of DNN models, n: total number of layers). For each scheduled layer X, the algorithm search for a layer Y scheduled later than layer X that can be scheduled at the completion time of layer X. If a layer Y is found, the algorithm re-order the layers to have layer Y right after layer X.

To identify layer Y, the post-processing algorithm searches for m (m: total number of DNN models) layers, which are head layers of each model at the completion time of layer X. This approach not only reduces the complexity but also ensure the layer dependence is not violated after re-ordering. Note that this approach is only possible on valid layer schedules based on valid layer order, which is obtained quickly by simple heuristics our scheduler exploits. We describe the post-processing algorithm in Figure 9.

E. Herald: An Implementation of the DSE algorithm

We codify the DSE algorithms we discussed in Subsection IV-C and Subsection IV-D and develop Herald. As illustrated in Figure 10, Herald receives user-selected dataflow styles for sub-accelerators and co-optimizes hardware resource distribution and layer execution schedule. Herald reports optimized PE and global NoC bandwidth partitioning with an optimized layer execution schedule for the partitioned sub-accelerators as outputs. Herald also reports estimated total latency and energy based on MAESTRO cost model [15] we extend for HDA use cases. Using Herald, we evaluate four example HDA architectures and identify one HDA architecture based on NVDLA [13] and Shi-diannao [12] dataflow styles that provide Pareto-optimal design points among various FDAs, RDAs, and scaled-out multi-DNN FDAs (SM-FDA) [24]. We discuss the evaluation we perform using Herald next.

V. Evaluations

To show the potential of HDAs, we evaluate four HDA designs with layer execution schedules generated by Herald using three workloads listed in Table II.

A. Evaluation Settings

Workloads. Based on AR/VR-motivated DNN models listed in Table I, we construct evaluation AR/VR workloads as listed in Table II. For each DNN model, we assign different numbers of batches to model different target processing rate of each

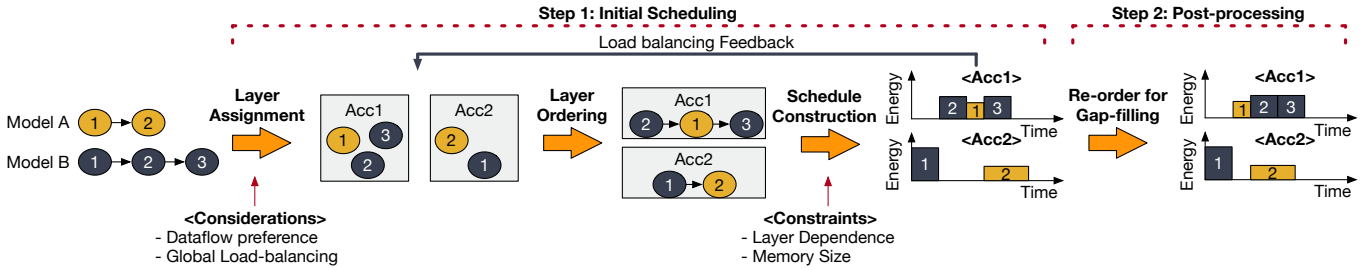

Fig. 7. An overview of layer scheduling algorithm of Herald. Circled numbers represent layers in each model.

```
Inputs
 - A list of hardware parameters of sub accelerators (Accs)
 - A list of DNN models to run, sorted in the dependence order (MD)
 - Load-balancing factor (LbF)
Outputs
- A list of (schedule time, layer ID, model), (Schedule)
- A list of completion time  for each sub-accelerator (Tot_Latency_Acc)

cycle = 0;
while MD.notEmpty do
  for model in MD do
    layer = model.head;
    // Get EDP/Latency for the layer on each acc
    (EDP, Latency) = MAESTRO_Herald.query(layer, Schedule, Accs) ;
    best_fit_acc = getAccIndex(min(EDP));
    // Check dependence, memory size, and load-balancing conditions
    dependence_cond = is_prev_layer_complelete(Schedule, model, cycle);
    mem_size_cond = MemorySize(cycle, Schedule) + cost.getMemSizeReq
                              < MemorySize;
    load_balance_cond = max(Tot_Latency_Acc)
                                < LbF * (Tot_Latency_Acc[acc] + Latency[best_fit_acc];

    if dependence_cond and mem_size_cond then
     if load_balance_cond then
       ToT_Latency_Acc[best_fit_acc] += Latency[best_fit_acc]; // Assign layer
       PopLayer(MD, layer);
       assigned_a_layer = true;
     else
       //Try the second, third, ... -best fit accelerator for load-balancing
     end if
    end if
    if assigned_a_layer then
     rearrange(MD); // Rearrange the order of model based on the layer ordering
                    //  strategy (depth-first, breadth-first. etc) selected by users
     break;
    end if
  end
  cycle = nextLayerCompletionTime(Schedule) // Failed to schedule; defer execution
end
```

Fig. 8. Layer assignment and ordering algorithm.

sub-task. In addition to the AR/VR workloads, we also evaluate ML-perf inference workload modeling multi-stream.

Dataflow. We combine two and three distinct dataflow styles from recent DNN accelerators(Shi-diannao [12], NVDLA [13]), and Eyeriss [14]. The selection of dataflow style is based on their distinct parallelization and data reuse strategies to maximize synergy. For example, Shi-diannao's dataflow parallelizes computations across output activation rows and columns and performs temporal accumulation of partial sums. In contrast, NVDLA's dataflow parallelizes computations across input and output channels and performs spatial accumulation of partial sums across input channels.

Combining dataflows with such different characteristics provide more dataflow flexibility than combining similar or the same dataflows. When we combine the same dataflow, we construct scaled-out multi-FDAs [24], which we also evaluate in Subsection V-B.

Cost Estimation. As we discussed in Subsection IV-B, we extend MAESTRO for the latency and energy estimation.

```
Inputs
 - A list of hardware parameters of sub accelerators (Accs)
 - A list of (schedule time, layer ID, model) (Schedule)
 - Look-ahead depth (LA)
Outputs
 - An updated schedule (Schedule)

for acc in ACCs do
 for baseLayerIdx in NumLayers(Schedule[acc]) do
  look-ahead =1
  while look-ahead  < LA do
    prev_completion_time = Schedule[acc][baseLayerIdx].completion_time
    test_layer = Schedule[acc][baseLayerIdx + look-ahead]
    // Test dependence, memory, load-balancing, and schedule overlap
    if layers is_schedulable(test_layer,  prev_completion_time, Schedule) then
      // Reorder the test layer
      UpdateSchedule(Schedule, test_layer, prev_completion_time)
    end if
  end
 end
end
```

Fig. 9. Post-processing algorithm that removes idle time based on bad layer execution order.

TABLE II
HETEROGENEOUS MULTI-DNN WORKLOADS USED FOR THE EVALUATION. WE MODEL AR/VR WORKLOADS USING MODELS LISTED IN TABLE I AND MLPERF [30].

Workload	Model	# of batches
AR/VR-A	Resnet50	2
	Unet	4
	MobileNetV2	4
AR/VR-B	Resnet50	2
	Unet	2
	MobileNetV2	4
	BR-Q Handpose	2
	Focal Length DepthNet	2
MLPerf	Resnet50	1 (and 8 for batch size study)
	MobileNetV1	1 (and 8 for batch size study)
	SSD-Resnet34	1 (and 8 for batch size study)
	SSD-MobileNetV1	1 (and 8 for batch size study)
	GNMT (RNN)	1 (and 8 for batch size study)

Accelerator Styles. For FDAs, we select NVDLA, Shi-diannao, and Eyeriss style accelerators. For RDAs, we select MAERI [18]. We run MAESTRO to analyze latency and CAD tools with a 28nm library to analyze energy. For HDAs, we select three two-way designs based on three dataflow styles selected for FDA and one three-way design combining all of those three dataflow styles.

We also model scale-out multi FDA (SM-FDA) [24] that scales out an FDA architecture within an accelerator chip. That is, sub-accelerators in an SM-FDA contain the same amount of hardware resources and run the same dataflow. We apply Herald's scheduler to show the pure impact of homogeneous dataflow and evenly-partitioned hardware resources.

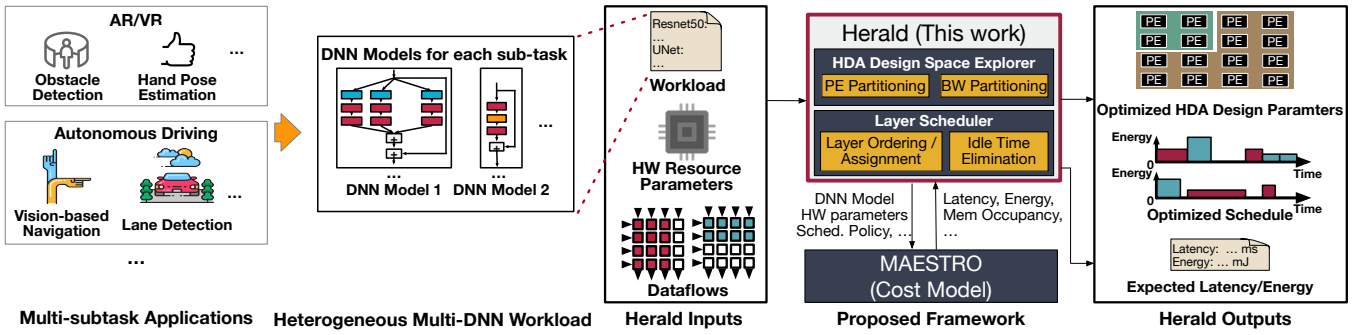

Fig. 10. Multi-DNN workloads from multi-subtask applications, which motivates heterogeneous DNN accelerators (HDAs). Targeting such workloads, we design an HDA optimization framework, Herald.

TABLE III
EVALUATED ACCELERATOR STYLES.

Accelerator Style	Dataflow
FDA	NVDLA
	Shi-diannao
	Eyeriss
Scaled-out Multi-FDA [24]	NVDLA + NVDLA
	Shi-diannao + Shi-diannao
	Eyeriss + Eyeriss
RDA	Flexible among three eval dataflows
HDA (This work)	NVDLA + Shi-diannao (**Maelstrom**)
	Shi-diannao + Eyeriss
	Eyeriss + NVDLA
	NVDLA + Shi-diannao + Eyeriss

TABLE IV
THREE ACCELERATOR CLASSES FOR EDGE, MOBILE, AND CLOUD USE SCENARIOS FOR EVALUATION.

Accelerator Class	Num. of PEs	NoC BW	Glob. Memory
Edge	1024	16 GB/s	4 MiB
Mobile	4096	64 GB/s	8 MiB
Cloud	16384	256 GB/s	16 MiB

Accelerators Classes. Based on previously proposed cloud and mobile accelerators [31], [32], we select the amount of hardware resources for edge, mobile, and cloud use scenarios as described in Table IV.

Schedulers. We apply the scheduling algorithm we discussed in Subsection IV-D in Herald. We also implement a baseline greedy scheduler and compare our scheduler against it.

B. Results

We highlight some observations that provide useful insights from our evaluation.

Costs and Benefits of HDAs. From Figure 11, we observe that well-optimized HDA and RDA design points are always on the Pareto curve over latency and energy, and FDA design points are not. On average, compared to the best FDA design with the lowest EDP, the best heterogeneous design provided 73.6% EDP improvements across all the case studies in Figure 11.

From the results in Figure 11, we identify that a two-way HDA architecture based on Shi-diannao and NVDLA dataflow styles provides the best latency and energy among four HDA designs we evaluate. We name the HDA design with optimized hardware partitioning identified by Herald as **Maelstrom**. We use Maelstrom as the reference HDA design for the rest of the evaluations.

TABLE V
MAELSTROM: OPTIMIZED HW RESOURCE PARTITION FOUND BY HERALD

Scenario	BW Partitioning (NVDLA / Shi)	PE Partitioning (NVDLA / Shi)
AR/VR-A, Edge	4 / 12	128 / 896
AR/VR-A, Mobile	40 / 24	1792 / 2304
AR/VR-A, Cloud	224 / 32	9728 / 6656
AR/VR-B, Edge	4 / 12	128 / 896
AR-VR-B, Mobile	48 / 16	1536 / 2560
AR/VR-B, Cloud	128 / 128	12032 / 4352
MLPerf, Edge	4 / 12	64 / 960
MLPerf, Mobile	32 / 32	1280 / 2816
MLPerf, Cloud	160 / 96	8192 / 8192

Maelstrom demonstrates 65.30% and 5.0% lower latency and energy compared than the best FDA, 63.11% and 4.1% lower latency and energy than the SM-FDAs [24], and 20.7% higher runtime but 22.0% lower energy compared to a MAERI-based RDA [18]. Such benefits are based on the synergy of NVDLA and Shi-diannao dataflow styles. When the number of channels is small, or the layer does not require accumulation across input channels (e.g., depth-wise CONV2D), NVDLA dataflow style significantly under-utilizes PEs. Shi-diannao provides high efficiency on such layers because of its parallelization strategy across output activation rows and columns. However, NVDLA provides higher efficiency for CONV2D layers with many channels and FC layers, which contains a large number of channels and performs accumulation across input channels.

Optimal HW Resource Partitioning. In Table V, we list the hardware resource partitioning results of Maelstrom design points with the best EDP for each workload and accelerator class. We observe that the optimal hardware partitioning is not trivial (e.g., evenly partitioned), which necessitates a systematic approach like Herald. This is because more number of active PEs requires more bandwidth, and the number of active PEs is a complex high-dimensional function of layer operation, layer size, number of PEs, mapping, and so on [15].

On average, across all the scenarios, we observe 111.12% more PEs are assigned to NVDLA style, which implies more number of layers in the workloads prefer NVDLA style than Shi-diannao style. However, on average, we observe 8.2% more bandwidth is assigned to Shi-diannao style, which shows higher

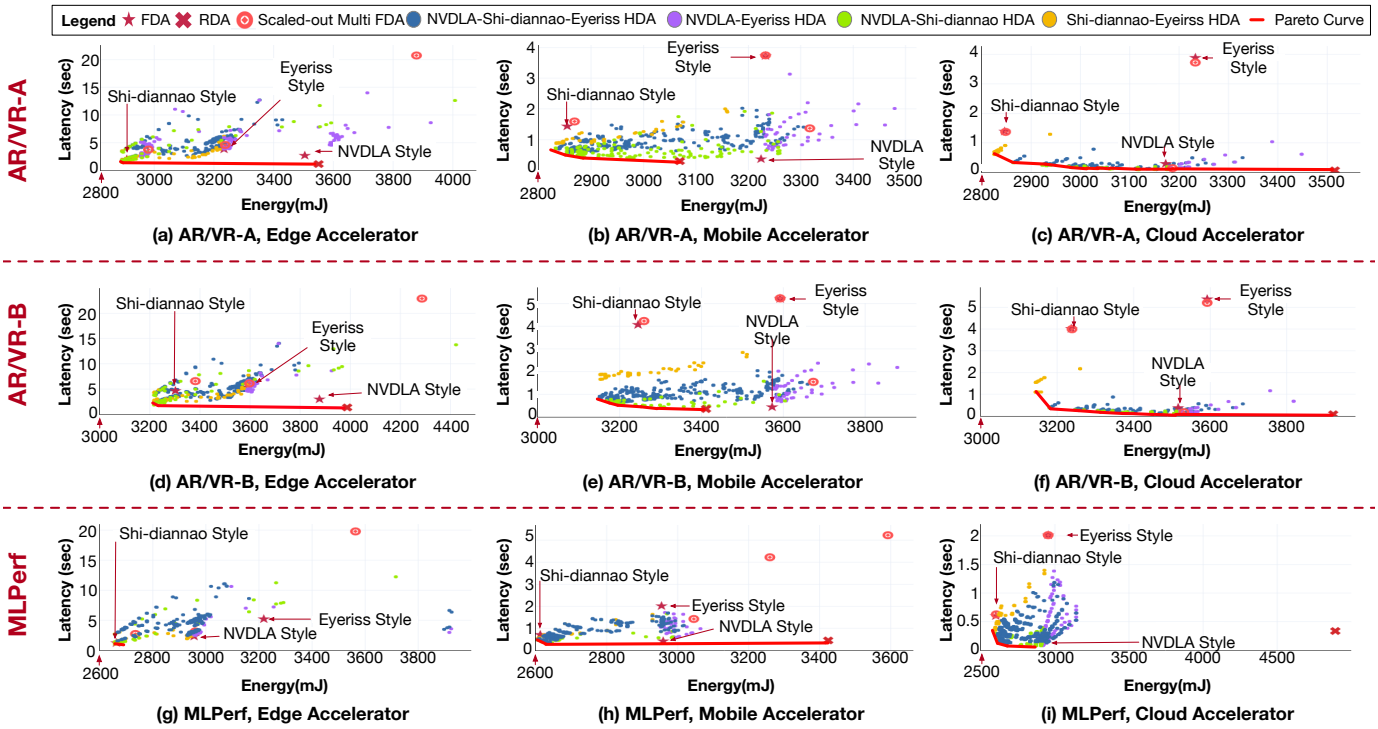

Fig. 11. Design space of two- and three-way HDAs. Each row of the plots shows results for each workload listed in Table II on three accelerator classes listed in Table IV. Each point in each plot represents a HW partitioning choice with an optimized schedule using Herald's scheduler. We label each FDA design point in each plot.

bandwidth requirements of Shi-diannao dataflow.

In particular, cloud accelerators have shown a stronger preference for NVDLA style dataflow, which resulted in 126.8% and 59.3% more bandwidth and PE assigned to NVDLA style sub-accelerator. This is related to the degree of parallelism each dataflow can exploit from the layers in the workload. NVDLA and Shi-diannao dataflows exploit channel and activation row/column parallelism, respectively. The maximum channel parallelism in the workload is 16.8M (FC layer 2, Focal Length DepthNet [17]), but the maximum activation parallelism in the workload is 334.1K (CONV layer 1, UNet [11]), which led to a stronger preference to NVDLA dataflow-style dataflow that exploits channel-parallelism. The maximum parallelization degree implies that we can design more powerful and efficient Maelstrom up to 16.8M PEs for three evaluated workloads until other conditions (e.g., memory, chip area, power, etc.) allow.

Although the workload is overall friendlier to NVDLA, the best designs of NVDLA-Shi-diannao HDAs, Maelstrom, balances between NVDLA and Shi-diannao styles, as shown in Table V. Such optimization results imply that the workload is highly heterogeneous, and Maelstrom successfully exploited its dataflow heterogeneity for the heterogeneous workload.

Impact of Workloads. Each row in Figure 11 shows the design space of three evaluation workloads, where we can observe the design space of HDAs depends on the workloads. In particular, we observe that workload with more heterogeneity and layers like AR/VR-B workload is more friendly to HDAs, providing

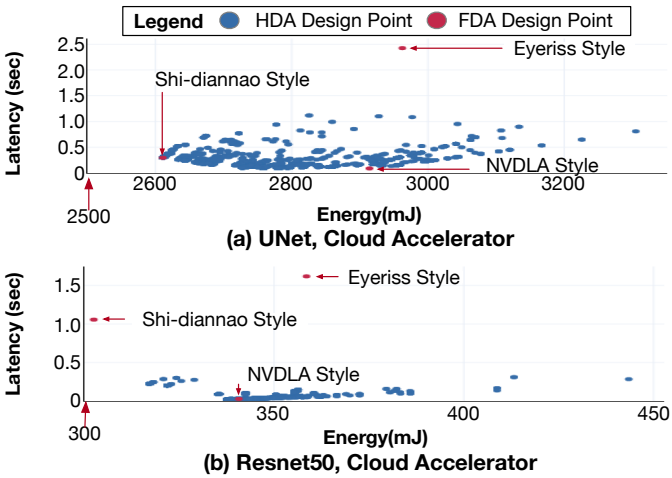

Fig. 12. Design space of single DNN use cases on (a) UNet and (b) Resnet50 based on cloud accelerator settings in Table IV.

86.8% latency and 6.61% energy improvements over bestFDAs for each case study in Figure 11, compared to 63.26% latency and 4.05% energy improvements for AR/VR-A and 48.1% latency and 4.4% energy improvements for MLPerf.

Single-DNN Case. Even for a single DNN, HDAs can still exploit layer parallelism and heterogeneity within a model by batch-processing the workload. We run UNet and Resnet50 using the batch size of four on the cloud scenario and present the results in Figure 12. We observe that the best FDA design is on the Pareto curve, unlike heterogeneous multi-DNN workloads we target. However, optimized Maelstrom designs

TABLE VI
LATENCY AND ENERGY GAIN AGAINST THE FDA AND RDA WITH THE BEST EDP ON VARIOUS BATCH SIZES ON MLPERF WORKLOAD.

Acc. Class	Batch Size	Latency Gain (vs FDA / vs RDA)	Energy Gain (vs FDA / vs RDA)
Edge	1	12.4% / -8.2%	0.2% / 20.4%
	8	21.28% / 26.7%	10.8% / 22.9%
Mobile	1	12.4% / -8.2%	0.2% / 17.1%
	8	56.0% / 76.1%	1.3% / 43.5%
Cloud	1	20.2% / 25.7%	10.8% / 26.8%
	8	63.9% / 80.4%	1.34% / 41.3%

still provide latency and energy benefits over monolithic designs. For UNet and Resnet50 workload, Maelstrom provided 26.4% and 48.1% EDP improvements over the best monolithic design. RDAs provided 22.5% and 29.0% lower latency compared to Maelstrom for UNet and Resnet50, respectively. However, RDAs required 11.7% and 15.8% more energy than Maelstrom for UNet and Resnet50, respectively.

Efficacy of Scheduling Algorithm. We compare the EDP of schedules from Herald's scheduler and a greedy scheduler, that assigns a sub-accelerator with the least EDP for each layer.. Compared to the greedy scheduler, Herald's scheduler considers global load balancing, exploits the layer dependence chain, and performs the post-processing discussed in Figure 7. On average, Herald's scheduler identified schedules on Maelstrom with 24.1% less EDP, compared to the greedy scheduler

Impact of Batch Size. We vary the batch size of MLPerf workload from one to eight and quantify the latency and energy benefits of HDAs. We summarize the latency and energy gain of HDAs in Table VI. We observe that HDA prefers large batch sizes, and HDA can outperform RDAs in both latency and energy when the batch size is large. On average, compared to RDAs, HDAs provided 3.1% latency and 21.4% energy savings on MLperf workload with batch size 1. When the batch size is increased to 8, HDAs provided 61.1% less latency and 35.9% less energy compared to RDAs, which shows HDA's preference for large batch sizes.

Comparison against RDAs. We evaluate a MAERI [18] style RDA and present the design points in Figure 11. Compared to Maelstrom in each scenario, RDA designs provided 22.9%, 21.5%, and 24.3% less latency for AR/VR-A, AR/VR-B, and MLPerf workloads, respectively. However, RDA designs required 18.7%, 15.5%, and 18.9% more energy for each workload, respectively. The extra energy cost of RDA is based on hardware components for reconfigurability. In contrast, an HDA can keep sub-accelerators with relatively simple architecture compared to flexible accelerators, which leads to better energy efficiency we present.

Results in Figure 11 show that both HDA and RDA architectures are Pareto-optimal. HDAs and RDAs have strength in energy and latency, respectively. The amount of benefits for latency and energy depends on the workload. Therefore, the choice of RDA or HDA depends on the performance goal, energy constraints, and the target workload.

Impact of Workload Change. Since DNN models evolve and applications change their inner implementation accordingly, workload change can occur after the deployment of an HDA.

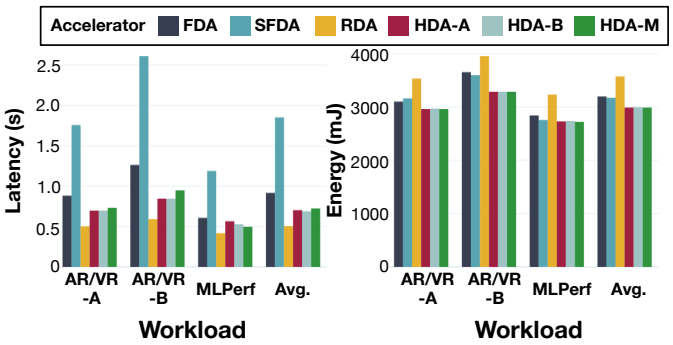

Fig. 13. Average latency and energy across edge, mobile, and cloud accelerator classes for each workload. HDA-A, HDA-B, and HDA-M refer to Maelstrom designs optimized for AR/VR-A, AR/VR-B, and MLPerf workload, respectively. SFDA refers to scaled-out FDA.

TABLE VII
AVERAGE TIME REQUIRED FOR SCHEDULING EACH WORKLOAD ON HDAS.

Workload	# Layers	# sub-accelerators	Scheduling Time (s)
AR/VR-A	448	2	2.89
		3	4.32
AR/VR-B	618	2	3.98
		3	10.74
MLPerf	181	2	1.61
		3	3.22

To deep-dive into the impact of such workload change after deployment of HDAs, we perform a case study of workload change in Figure 13. In the case study, we fix each HDA design and only perform layer execution scheduling for running different workloads than a workload each HDA design is optimized for (e.g., running AR/VR-B workload on an HDA originally optimized for AR/VR-A workload).

From the case study, we observe that running different workloads than the workload each HDA is optimized for results in minor latency and energy increase, 4.0% and 0.1%, on average. On average, across all the workload change scenarios, HDAs provided 19.44% energy savings at the cost of 28.6% latency against RDAs. Against FDAs, HDAs provided 29.99% latency and 6.45% energy savings, on average.

Scheduling Time. Although Herald's scheduler is designed to be offline, the scheduler is light-weighted. We run Herald on a laptop with i9-9880H processor and 16GB memory and present the time required for scheduling on each HDA design point in Table VII, since overall execution time heavily depends on user parameters (e.g., search granularity, number of sub-accelerators, etc.). On average, the scheduling requires 11.09 ms per layer and per HDA design point.

Summary. We summarize our main observations below:

- The design space of HDA is not trivial, which requires a systematic co-optimization of hardware resource partitioning and layer execution schedule.
- We identify a promising HDA architecture Maelstrom based on NVDLA and Shi-diannao dataflow styles. Maelstrom designs outperform FDA and SM-FDA designs in overall latency and energy.
- Both Maelstrom and RDA architectures are Pareto-optimal

design points with different strengths in energy efficiency and latency, respectively.

- Simple combination of sub-accelerators running the same dataflow does not lead to Pareto-optimal design points, which shows the efficacy of HDAs.

VI. Related Works

DNN Dataflows and Accelerators. Shi-diannao [12] is an FDA designed to be embedded near sensors, which exploits convolutional reuse via an output-stationary style dataflow. Eyeriss [27] is one of the state-of-the-art low-energy DNN accelerators that introduced dataflow taxonomy and a new dataflow style, row-stationary. Fused-layer CNN accelerator [33] exploited fine-grained pipelined layer parallelism that minimizes activation data movement. Flexflow [21] is an RDA that supports three distinct dataflows. Tensor Processing Unit(TPU) [31] is a systolic array-based DNN accelerator designed for cloud workload in data centers. MAERI [18] is an RDA that efficiently supports irregular mappings resulting from sparsity, cross-layer mapping [33], and so on. Tangram [34] is a DNN accelerator that explored pipelined layer parallelism within a model with optimized dataflow for such back-to-back layer execution. Interstellar [20] presented the importance of loop blocking (tile sizing) in DNN accelerators utilizing Halide [35]. Prema [36] explored the use of preemption scheduler implemented in hardware for QoS of multiple-DNNs, which targets a systolic array-based FDA.

Multi-DNN Accelerators. Shen et al. explored the use of multiple FDA sub-accelerators running the same dataflow termed as convolutional layer processors in FPGAs [37]. AI-multitasking architecture [24] employed multiple systolic arrays within an accelerator chip and parallelized computation tiles of each layer. Although the idea of employing sub-accelerators is proposed in those works, *Herald first explored the dataflow heterogeneity using sub-accelerators, co-optimizing hardware resource partitioning optimization and layer execution schedule.*

Heterogeneous Accelerators. Chandramoorthy et al. [38] explored accelerator-rich chip-multiprocessor that include various accelerators for different tasks in image processing. Although the work included a convolution module among sub-accelerators, the convolution module provides only one dataflow style, focusing on general image kernels, not DNNs. Master of None Acceleration [39] explored a heterogeneous accelerator for analytical query and presented that the design space of heterogeneous accelerators for the target domain has both beneficial and disadvantageous design points.

VII. Conclusion

In this paper, we explored the latency and energy optimization opportunities of heterogeneous dataflow accelerators (HDAs) on recent heterogeneous multi-DNN workloads such as AR/VR. Because the efficiency of a DNN accelerator depends on mapping, workload, and hardware design parameters at the same time, identifying the best HDA design point with an optimized schedule is challenging. Therefore, we developed Herald, an automated co-design space exploration framework for hardware resource partitioning and layer scheduling for heterogeneous DNN accelerators. In our evaluation, Herald identified a promising HDA architecture, Maelstrom, which deploys NVDLA and Shi-diannao dataflow styles over two sub-accelerators. Maelstrom provided, on average, 73.6% EDP benefits compared to the best fixed dataflow accelerator designs we compare across three workloads we evaluate. We observe that the most efficient Maelstrom design points have non-trivial hardware resource partitioning, and a naive scheduler can result in EDP degradation, motivating a systematic approach like Herald.

In summary, HDA is a new promising class of flexible dataflow accelerators, and Herald facilitates the design of HDAs via co-optimization of hardware resource partitioning across sub-accelerators and layer execution schedule.

Acknowledgements

We thank Simon Hollis, Meng Li, Pierce Chuang, Ganesh Venkatesh, and Yilei Li for insightful comments and discussions. This work was supported in-part by NSF Award OAC-1909900.

References

[1] C.-J. Wu, D. Brooks, K. Chen, D. Chen, S. Choudhury, M. Dukhan, K. Hazelwood, E. Isaac, Y. Jia, B. Jia *et al.*, "Machine learning at facebook: Understanding inference at the edge," in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2019, pp. 331–344.

[2] Y. Tian, K. Pei, S. Jana, and B. Ray, "Deeptest: Automated testing of deep-neural-network-driven autonomous cars," in *Proceedings of the 40th international conference on software engineering*, 2018, pp. 303–314.

[3] S. Lee, S. W. Oh, D. Won, and S. J. Kim, "Copy-and-paste networks for deep video inpainting," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 4413–4421.

[4] V. Ramanishka, Y.-T. Chen, T. Misu, and K. Saenko, "Toward driving scene understanding: A dataset for learning driver behavior and causal reasoning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 7699–7707.

[5] K. Hazelwood, S. Bird, D. Brooks, S. Chintala, U. Diril, D. Dzhulgakov, M. Fawzy, B. Jia, Y. Jia, A. Kalro *et al.*, "Applied machine learning at facebook: A datacenter infrastructure perspective," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2018, pp. 620–629.

[6] S. Rabii, E. Beigne, V. Chandra, B. D. Salvo, R. Ho, and R. Pendse, "Computational directions for augmented reality systems," in *2019 IEEE Symposium on VLSI Circuits, plenary*, 2019.

[7] A. S. Kaplanyan, A. Sochenov, T. Leimkühler, M. Okunev, T. Goodall, and G. Rufo, "Deepfovea: Neural reconstruction for foveated rendering and video compression using learned statistics of natural videos," *ACM Transactions on Graphics (TOG)*, vol. 38, no. 6, pp. 1–13, 2019.

[8] M. Abrash, "Inventing the future," https://www.oculus.com/blog/inventing-the-future, 2017.

[9] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[10] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," *arXiv preprint arXiv:1801.04381*, 2019.

[11] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical image computing and computer-assisted intervention*. Springer, 2015, pp. 234–241.

[12] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam, "Shidiannao: Shifting vision processing closer to the sensor," in *International Symposium on Computer Architecture (ISCA)*, 2015.

[13] NVIDIA, "Nvdla deep learning accelerator," http://nvdla.org, 2017.

[14] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," in *International Symposium on Computer Architecture (ISCA)*, 2016.

[15] 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 *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 754–768.

[16] M. Madadi, S. Escalera, X. Baró, and J. Gonzalez, "End-to-end global to local cnn learning for hand pose recovery in depth data," *arXiv preprint arXiv:1705.09606*, 2017.

[17] L. He, G. Wang, and Z. Hu, "Learning depth from single images with deep neural network embedding focal length," *IEEE Transactions on Image Processing*, vol. 27, no. 9, pp. 4676–4689, 2018.

[18] H. Kwon, A. Samajdar, and T. Krishna, "Maeri: Enabling flexible dataflow mapping over dnn accelerators via reconfigurable interconnects," in *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2018, pp. 461–475.

[19] A. Parashar, P. Raina, Y. S. Shao, Y.-H. Chen, V. A. Ying, A. Mukkara, R. Venkatesan, B. Khailany, S. W. Keckler, and J. Emer, "Timeloop: A systematic approach to dnn accelerator evaluation," in *2019 IEEE international symposium on performance analysis of systems and software (ISPASS)*. IEEE, 2019, pp. 304–315.

[20] X. Yang, M. Gao, Q. Liu, J. Setter, J. Pu, A. Nayak, S. Bell, K. Cao, H. Ha, P. Raina *et al.*, "Interstellar: Using halide's scheduling language to analyze dnn accelerators," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020, pp. 369–383.

[21] W. Lu, G. Yan, J. Li, S. Gong, Y. Han, and X. Li, "Flexflow: A flexible dataflow accelerator architecture for convolutional neural networks," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2017, pp. 553–564.

[22] Y.-H. Chen, T.-J. Yang, J. Emer, and V. Sze, "Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 2, pp. 292–308, 2019.

[23] Z. Zhao, H. Kwon, S. Kuhar, W. Sheng, Z. Mao, and T. Krishna, "mrna: Enabling efficient mapping space exploration for a reconfiguration neural accelerator," in *2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2019, pp. 282–292.

[24] E. Baek, D. Kwon, and J. Kim, "A multi-neural network acceleration architecture," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020, pp. 940–953.

[25] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 815–823.

[26] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2961–2969.

[32] Qualcomm, "Quacomm hexagon 680," https://www.hotchips.org/wp-content/uploads/hc_archives/hc27/HC27.24-Monday-Epub/HC27.24.20-Multimedia-Epub/HC27.24.211-Hexagon680-Codrescu-Qualcomm.pdf, 2015.

[27] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, 2016.

[28] H. Kwon, P. Chatarasi, M. Pellauer, A. Parashar, V. Sarkar, and T. Krishna, "Maestro: A data-centric approach to understand reuse, performance, and hardware cost of dnn mappings," *IEEE Micro*, vol. 40, no. 3, pp. 20–29, 2020.

[29] R. Guirado, H. Kwon, E. Alarcón, S. Abadal, and T. Krishna, "Understanding the impact of on-chip communication on dnn accelerator performance," in *2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*. IEEE, 2019, pp. 85–88.

[30] P. Mattson, V. J. Reddi, C. Cheng, C. Coleman, G. Diamos, D. Kanter, P. Micikevicius, D. Patterson, G. Schmuelling, H. Tang *et al.*, "Mlperf: An industry standard benchmark suite for machine learning performance," *IEEE Micro*, vol. 40, no. 2, pp. 8–16, 2020.

[31] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *International Symposium on Computer Architecture (ISCA)*. IEEE, 2017, pp. 1–12.

[33] M. Alwani, H. Chen, M. Ferdman, and P. Milder, "Fused-layer cnn accelerators," in *The 49th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Press, 2016, p. 22.

[34] M. Gao, X. Yang, J. Pu, M. Horowitz, and C. Kozyrakis, "Tangram: Optimized coarse-grained dataflow for scalable nn accelerators," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 2019, pp. 807–820.

[35] J. Ragan-Kelley, C. Barnes, A. Adams, S. Paris, F. Durand, and S. Amarasinghe, "Halide: a language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines," *Acm Sigplan Notices*, vol. 48, no. 6, pp. 519–530, 2013.

[36] Y. Choi and M. Rhu, "Prema: A predictive multi-task scheduling algorithm for preemptible neural processing units," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2020, pp. 220–233.

[37] Y. Shen, M. Ferdman, and P. Milder, "Maximizing cnn accelerator efficiency through resource partitioning," in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2017, pp. 535–547.

[38] N. Chandramoorthy, G. Tagliavini, K. Irick, A. Pullini, S. Advani, S. Al Habsi, M. Cotter, J. Sampson, V. Narayanan, and L. Benini, "Exploring architectural heterogeneity in intelligent vision systems," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2015, pp. 1–12.

[39] A. Lottarini, J. P. Cerqueira, T. J. Repetti, S. A. Edwards, K. A. Ross, M. Seok, and M. A. Kim, "Master of none acceleration: a comparison of accelerator architectures for analytical query processing," in *Proceedings of the 46th International Symposium on Computer Architecture*. ACM, 2019, pp. 762–773.