HERALD: OPTIMIZING HETEROGENEOUS DNN ACCELERATORS FOR EDGE DEVICES

Hyoukjun Kwon\textsuperscript{1,2} Liangzhen Lai\textsuperscript{2} Tushar Krishna\textsuperscript{1} Vikas Chandra\textsuperscript{2}

Abstract
Recent advances in deep neural networks (DNNs) have made DNNs the backbone of many applications on edge devices such as face recognition, object detection, and so on. To deal with massive computation requirements of DNN inferences within stringent energy and latency constraints, DNN accelerators (i.e., hardware specialized for DNN inferences), have emerged as a promising solution. Such advancement of hardware supporting DNNs has led to multiple DNN-based applications running at the same time on edge devices. They often run in parallel as background processes or as sub-tasks of a complex application. Thus, DNN workloads on a DNN accelerator now include a variety of layer operations and sizes from DNN models for diverse applications, which makes them heterogeneous in layer granularity. Such heterogeneous workloads introduce a new major challenge for monolithic DNN accelerators because the efficiency of DNN accelerators relies on its dataflow, and different DNN layer types and shapes prefer different dataflows.

In this work, we propose to tackle this challenge by designing heterogeneous DNN accelerators (HDAs) that deploy multiple DNN accelerators each optimized for different layer shapes and operations. To enable this approach, we propose HERALD, an optimization framework that explores the design space an HDA and layer execution schedules. HERALD mainly targets design time, but HERALD can perform scheduling at runtime when the target workload changes after deployment of an HDA. Design time-optimized HDAs with the best energy-delay-product (EDP) HERALD identified provided 24.93% EDP benefits with 16.1% latency and 7.6% energy benefits on average across workloads and accelerators we evaluate compared to the best case of monolithic DNN accelerators for each evaluation setting by deploying two complementary-style DNN accelerators. HERALD’s scheduler employs heuristics that exploit the characteristics of DNN workloads, which provided 6.4% better EDP on average compared to a baseline scheduler.

1 Introduction
Enhanced deep neural network (DNN) computation capability on edge devices such as smartphones and AR glasses enables the deployment of diverse applications that heavily rely on DNNs, such as face recognition (Taigman et al., 2014), image super-resolution (Park et al., 2018), and so on. The improvements in mobile compute units including CPUs, GPUs, DSPs, and NPUs (neural processing units) have provided computation capabilities for real-time DNN inference but with a reduced processing rate (less than 30fps) for a single task using single DNN model (Hazelwood et al., 2018; Li et al., 2018). Moreover, DNN accelerators (i.e., hardware specialized in DNN inference computation) provide massive improvements in performance and energy efficiency of DNNs. Therefore, edge devices have started to employ DNN accelerators (Apple, 2017; Huawei, 2018; 2019; Qualcomm, 2018) to run DNN-based applications under edge devices’ stringent energy and latency constraint.

Such innovations in hardware supporting DNNs has led to wider adoption of DNNs in edge device applications. Some applications have DNNs running continuously in the background (e.g., keyword detection), while some others employ multiple DNNs for sub-tasks. For example, an AR application can require object detection, hand tracking, speech recognition, pose estimation, and so on (Abrash, 2017; Rabii et al., 2019), which all rely on different DNN models for getting state-of-the-art performance. Therefore, the DNN workload from various applications for a DNN accelerator is expected to be heavier and more diverse compared to workloads (Ignatov et al., 2018) that ran only one model at each time. In other words, a DNN accelerator will need to run multiple distinct models at the same time, and the models include diverse layer operations and sizes, which makes the overall DNN workload on an accelerator heterogeneous.

Such heterogeneous workloads impose a new challenge
Figure 1. EDP estimation of Shidiannao (Du et al., 2015), NVDLA (NVIDIA, 2017), and Eyeriss’s row-stationary (Chen et al., 2016a) style dataflows on Resnet50, MobileNetV2, and UNet. We apply 256 PEs and 32GBps NoC bandwidth to MAESTRO cost model (Kwon et al., 2018a) to estimate the energy and latency. The break-down is in the granularity of unit operations.

for designing DNN accelerators with stringent energy, latency and quality of service constraints imposed by edge devices. The source of the challenge is that DNN accelerator dataflows are often over-specialized for specific layer shapes and operations - which is what provides them an efficiency boost in the first place. However, the same design-paradigm leads to huge inefficiency drops for non-preferred workloads (Kwon et al., 2018a). Also, tuning an accelerator for the average case can lead to uniform inefficiency across all the layers in heterogeneous DNN workloads, as Figure 1 implies. For example, NVDLA (NVIDIA, 2017) style accelerators exploit input and output channel parallelism, which provides near roof-line throughput for CONV2D layers with deep channels as shown in Figure 1 (a) and (b). However, when it runs a CONV2D layer with a small number of channels, NVDLA style accelerator suffers severe compute unit under-utilization, which leads to low throughput and energy efficiency, as results in Figure 1 (c) implies.

Such efficiency drop based on layer heterogeneity in operation and shape can be significant based on the target DNN models. For example, in a recent classification network, Resnet50 (He et al., 2016), layers with deep channels and low activation resolution1 are dominant; all the layers except the first layer are such layers. Therefore, NVDLA style accelerators provide high compute unit utilization all the layers except the input layer. However, in a segmentation network, UNet (Ronneberger et al., 2015), layers with shallow channels and high activation resolution account for 41.7% of the layers. NVDLA style accelerators suffer from low compute unit utilization on those less favorable layers, resulting in severe efficiency drop; 3.6× slower than Shi-diannao style while NVDLA style is 3.0× faster than Shi-diannao style on Resnet50 (Kwon et al., 2018a). Flexible accelerators (Kwon et al., 2018b; Lu et al., 2017) are potential solutions to solve the workload heterogeneity problems. But the reconfigurability can cause static efficiency overheads, which makes them less desired for edge devices.

To deal with such challenges, we propose a heterogeneous DNN accelerator (HDA) that contains diverse sub-accelerators within an accelerator chip. This can be a promising option since we can cherry-pick the most efficient sub-accelerator for each layer. Also, by having multiple sub-accelerators, we can exploit layer parallelism, which can be extremely useful for edge device workloads with many DNNs running simultaneously. However, because we split finite hardware resources within a chip to multiple sub-accelerators, each sub-accelerator may be less powerful compared to a full homogeneous (or, monolithic) accelerator. Therefore, as reported in a recent heterogeneous accelerator work for database query (Lottarini et al., 2019), heterogeneous accelerators can be either more efficient or inefficient than monolithic accelerators based on the design parameters (architecture), compiler mapping (dataflow), and workload (models). We observed the same trend for DNNs as well as presented in Section 5.2, which implies two findings: (1) Heterogeneous accelerators have potential latency and energy gains over monolithic accelerators (2) To materialize such benefits, we need a systematic optimization framework that considers all of the aforementioned three aspects: architecture, dataflow, and DNN models.

Therefore, we propose an HDA optimization framework, HERALD, illustrated in Figure 2. As inputs, HERALD receives total amounts of available hardware resources (number of PEs, NoC bandwidth, and so on), dataflows of sub-accelerators, and target DNN models to run. As outputs, HERALD generates hardware resource partitioning for each sub-accelerator, layer execution schedule, and estimated total latency and energy using the MAESTRO analytic model (Kwon et al., 2018a). That is, HERALD is an HDA design space explorer and layer-granularity scheduler for user-specified workloads. HERALD mainly targets design time to fully exploit the benefits of specialization on a target workload, identifying optimized hardware configuration and schedule for the workload. When the workload changes at run time, HERALD can still generate an optimized schedule for the new workload for the underlying hardware. In our evaluations, we use HERALD to create HDA design points combining multiple accelerators with complementary dataflow styles. The design points with the best EDP for each experiment provided 24.9% smaller

---

1 we compare the number of input channels (C) and input activation height (Y) to define deep/shallow channels and high/low activation resolution; if C > Y, deep and low-res., and vice versa.
Figure 3. An illustration of fundamental convolution operation (CONV2D), (a) shows a visualization of the operation as a sliding window stencil operation, and (b) shows the precise description of CONV2D in a loop nest. Note that we illustrate one representative way (dataflow) to compute CONV2D and do not include non-linear functions in the illustration for simplicity.

energy-delay product (16.1% latency and 7.6% energy benefits) across complex edge device workloads we evaluate compared to the best monolithic design we evaluate.

We summarize the contribution of this paper as follows:

1. To the best of our knowledge, HERALD is the first work that explored heterogeneous architecture and layer scheduling in DNN accelerator domain.
2. HERALD automatically searches for optimal hardware resource partitioning among sub- accelerators in HDAs.
3. HERALD explores optimal layer schedules that match layers with the most efficient accelerator for each considering both of local (layer- accelerator matching) and global (load-balancing) optimization.

2 BACKGROUND

2.1 DNN operations and layer sizes

Many DNNs such as convolutional neural networks (CNNs), recurrent neural networks (RNNs), spiking neural networks (SNNs) have been proposed. Among them, CNNs are popular for computer vision applications, which is dominant in edge device applications. Therefore, we focus on CNNs and operations in recent CNN models as listed in Table 2.

The basis of CNN operation is convolutions (CONV2D), a six-dimensional (or seven if include batch) multiplication and addition (MAC) operations over two tensors; input activation and filter weights, as illustrated in Figure 3. Other than CONV2D, many recent DNN operators are variants of the CONV2D operation. Some of the operators such as depth-wise separable convolution consist of multiple sub-operators or layers; depth-wise and point-wise convolutions. Since we explore optimization opportunities in layer granularity, we focus on each unit operators. Figure 4 illustrates some representative unit operations. We also list those unit operators and highlight their features in data reuse in Table 1 since data reuse is the key component for energy cost.

2.2 Monolithic and Heterogeneous DNN Accelerators

DNN accelerators are hardware specialized for DNN forward (or, both of forward/backward) pass, which provides tremendous throughput and energy benefits over CPUs and GPUs. For example, TPU (Jouppi et al., 2017) reported 53.5× and 25.74× higher performance of two CNNs over CPUs and GPUs on average, respectively. To achieve such a high efficiency, DNN accelerators exploit hundreds of processing elements to deal with billions of multiply-and-accumulate (MAC) operations in DNN models and exploit scratchpad memory and inter-PE interconnection network to maximize data reuse to deal with massive energy cost for fetching data from DRAM.

Most of previously proposed DNN accelerators are homogeneous, or monolithic, which has regular architecture as shown in Figure 5 (a) and runs only one dataflow style. Monolithic DNN accelerators can have more complicated
3 Motivation

In this section, we discuss the heterogeneity of edge device workloads and the impact of dataflow style on the efficiency of a DNN accelerator, which motivates HDAs.

3.1 Layer Heterogeneity in Recent DNN Models

From recent DNN models, we can observe two classes of heterogeneity: layer shape (or, size of each layer dimension) and layer operations.

3.1.1 Layer Shape

Classification networks such as Resnet (He et al., 2016) 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 tend to 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 7 (a).

In contrast, segmentation networks such as UNet (Ronneberger et al., 2015) 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 methods such as transposed convolution (a.k.a. deconvolution). As a result, layer shapes in segmentation networks follow the trend illustrated in Figure 7 (b).

3.1.2 Layer Operation

In Figure 8 (a) and (b), we illustrate the structure of two recent DNN models, ResNeXt (Saining Xie & He, 2017) and MobileNetV2 (Mark Sandler & Chen, 2019), respec-
Figure 8. Layer operations in ResNeXt (Saining Xie & He, 2017) and MobileNetV2 (Mark Sandler & Chen, 2019). P on CONV2D layers represent point-wise convolution (1x1 kernel).

Figure 9. The impact of dataflow, layer operation, and layer shape on the efficiency of a DNN accelerator. We compare to distinct dataflow and its supporting architecture. Gray-colored compute units represent underutilized compute units. Red-colored compute units represent underutilized compute units. Blue-colored compute units represent underutilized compute units. Green-colored compute units represent underutilized compute units. Yellow-colored compute units represent underutilized compute units. Purple-colored compute units represent underutilized compute units.

3.1.3 DNN Models on Edge Devices

We list up lists up recent DNN models related to applications on edge devices in Table 2. Based on tasks, layer shapes and operations are diverse. Because of diverse sub-tasks in recent edge-device applications, DNN models used in edge device applications are heterogeneous in layer shape and operation as listed in Table 1. From those two examples, we can observe the diversity of layer operations.

3.2 Dataflow and Efficiency of DNN Accelerators

Dataflows and architectures to support the dataflow significantly affect the efficiency of a DNN accelerator with distinct preference to layer shapes and operation. We show examples of such cases in Figure 9, comparing Shidiannao (Du et al., 2015) and NVDLA (NVIDIA, 2017) style accelerators. Those two accelerators have distinct approaches to compute MAC operations in DNNs. Shidiannao style parallelizes activation width and height using an output-stationary style dataflow, which exploits convolutional reuse across kernels. NVDLA style parallelizes input and output channels using a weight-stationary style dataflow, which exploits activation reuse across output channels. Their parallelization strategies result in dramatically different utilization of compute units, as shown in Figure 9. In addition to the mapping utilization, each of dataflow style 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 (Kwon et al., 2018a).

We show overall efficiency in EDP of three dataflow styles on ResNet50 (He et al., 2016), MobileNetV2 (Mark Sandler & Chen, 2019), and U-Net (Ronneberger et al., 2015) in Figure 9, considering all of those aspects. Note that the breakdown is in a fine-grained unit operation (e.g., a linear bottleneck in MobileNetV2 is divided into two point-wise and one depth-wise convolution). Because NVDLA-style dataflow prefers late layers with many channels, it out-performs the other two dataflows on ResNet50 and MobileNetV2, which are dominated by late layers. Similarly, because the other two styles prefer early layers, they out-perform NVGLA style on UNet, which includes many early layers. Even among late layers, the efficiency differs based on each layer’s actual shape; late layers in UNet have higher resolution than those in the other two models, so Shi-diannao and row-stationary styles also perform well.

Therefore, no single dataflow style is good for all the layers, and we need to optimize the dataflow for our target workload. However, when the target workload is heterogeneous, the common practice to optimize the dataflow for the average case of the workload can result in a consistently inefficient dataflow for all the layers in the workload, which is a major challenge for DNN acceleration in edge devices.

3.3 Benefits of HDAs

In edge devices that run multiple and heterogeneous DNN models for multiple DNN-based applications, efficiently supporting such heterogeneous workloads using a monolithic DNN accelerator is challenging because optimizing for the average case can make the DNN accelerator uniformly inefficient for highly heterogeneous workloads, as discussed in Section 3.2. In contrast, HDAs can provide better efficiency from the following aspects.

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

Latency hiding via layer parallelism. Unlike most of the monolithic accelerators run one layer and another, HDAs can run multiple layers of different models on each sub-accelerator in parallel. By running multiple layers in parallel, a heterogeneous accelerator can overlap the latency of multiple models, which leads to latency hiding among DNN models reducing overall latency.
Table 2. DNN models selected for evaluation. For works without model name, we name them to refer to those works in the rest of paper.

| Task                     | Model               | Layer Shape | Layer Operations                  |
|--------------------------|---------------------|-------------|-----------------------------------|
| Image Classification     | ResNet50 (He et al., 2016) | Classification | CONV2D, FC, Skip-Con.         |
| Image Classification     | MobileNetV2 (Mark Sandler & Chen, 2019) | Classification | CONV2D, DWCONV, Skip-Con.     |
| Image Segmentation       | UNet (Ronneberger et al., 2015) | Segmentation | CONV2D, FC, TRCONV, Concat.  |
| Depth Estimation         | Focal Length DepthNet (He et al., 2018) | Segmentation | CONV2D, FC, UPCONV              |
| Hand Pose Estimation     | Br-Q HandposeNet (Madadi et al., 2017) | Classification | CONV2D, FC                  |

However, because each sub-accelerator contains smaller amount of hardware resources than a monolithic DNN accelerator under the same overall hardware resources, HDAs need to be carefully designed. Also, because of multiple sub-accelerators, a scheduler needs to search optimal layer execution schedules on an HDA, which can determine the overall efficiency.

Therefore, we developed HERALD, an HDA optimization framework that contains a hardware design space explorer, and a layer scheduler. Exploiting those sub-components, HERALD automates HDA design tailored for user-specified target models and outputs estimated latency and energy using the optimized design. We discuss details of HERALD next.

4 HERALD FRAMEWORK

4.1 Execution Model and Workloads

In this work, we target layer granularity execution on each sub-accelerator of HDAs because we observe significant dataflow preference of layers (Kwon et al., 2018a; Lu et al., 2017) and more fine-grained scheduling results in high control and scheduling overhead. We assume the following execution steps in HERALD:

1. Fetch filter values from DRAM and store them in a global buffer.
2. Distribute filter values to sub-accelerators based layer execution schedule.
3. Fetch activation from DRAM and store them in the global buffer.
4. Stream activation values to their corresponding sub-accelerators based on layer execution schedule.
5. Store streamed-out output activation from each sub-accelerator to the global buffer.
6. 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.

For steps 3 and 6, activation is stored in DRAM and loaded in a tiled manner specified by the dataflow in target accelerator if the buffer size is not sufficient to store entire activation. When output activation is committed to the global buffer, HERALD in 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 layout. When the data layout and miscellaneous context change overheads, HERALD also provides an option to specify the latency and energy penalties for them.

4.2 Accelerator Design Space Exploration

As shown in Figure 2, HERALD receives the total amount of PEs, memory size, and memory/NoC bandwidth as inputs. Unlike monolithic DNN accelerators, HDAs need to distribute such resources for each sub-accelerator. However, evenly distributing those resources does not yield the most optimal HDAs because each accelerator’s dataflow style has a different balance between the number of PEs, memory size, and bandwidth requirements as discussed in Section 3.2. Therefore, HERALD explores the resource partitioning space for each resource type, which constructs a nested combinatorial optimization problem, or a nested resource partitioning problem.

We implement a combinatorial optimization framework upon a behavioral analytic cost model for DNN acceler-
ators, MAESTRO (Kwon et al., 2018a), which estimates the latency and energy for input layers, dataflows, and hardware parameters (number of PEs, NoC bandwidth, etc.). In Figure 10, we show an example design space from PE partitioning on two sub-accelerators in a 16K-PE-HDA fixing other design parameters assuming even bandwidth partitioning (128/128 GBps). We can observe that the design space has irregular cost variations so evenly partitioning (8K/8K PE partitioning) does not yield the most efficient HDA.

Based on user-specified framework options, HERALD’s design space exploration (DSE) tool either performs an exhaustive search, binary sampling-based search, or random search. Binary sampling-based search first evaluates 2^n design points with a regular interval with user-specified parameter n. Afterward, HERALD selects an interval between two adjacent evaluated design points with the lowest average cost in energy-delay product and performs an exhaustive search over the selected interval. The random search follows a similar approach as the binary-sampling-based search but selects random pre-evaluated design points.

4.3 Layer Scheduling

The goal of scheduling in HERALD is to minimize the energy consumption and latency of an HDA, exploiting different preferences of each layer to accelerators. However, the layer scheduling space is massive. For example, 2.54 × 10^21 possible layer execution schedules exist for workload A in Table 3 even if we only consider permutation of the layers on a single accelerator. To deal with such a large search space, we develop a heuristic the characteristics of DNN workloads to reduce the scheduling overhead. Two major characteristics we exploit are the dependence among layers; layers have linear dependence chain in most models, and they are dependent across models. In Figure 11, we present an overview of the layer scheduling algorithm in HERALD that consists of three steps: layer assignment to each sub-accelerator, layer ordering within each sub-accelerator, and timeline construction.

Layer Assignment. HERALD first assigns layers to each sub-accelerator without considering their execution order. HERALD implements latency-, energy-, and EDP-greedy methods with global load-balancing. That is, HERALD first assigns a layer to a sub-accelerator that provides the lowest cost and re-assigns the layer to the second most efficient sub-accelerator if the cost increase is negligible and the original assignment introduces unbalanced-load, which actually degrades the overall efficiency. The threshold cost increase in percentage and the degree of load-unbalance to trigger the layer re-assignment are user-configurable parameters. As discussed, the layer assignment algorithm in HERALD’s scheduler balances local optimization via greedy-method and global optimization via load-balancing consideration.

Layer Ordering. Once all the layers are assigned on sub-accelerators, HERALD determines the order of layer execution in each sub-accelerator. HERALD implements two layer ordering methods, depth-first and breadth-first, as an example in Figure 12 shows. Depth-first layer ordering first schedules all the layers in the first model of the workload, and then moves on to the next model, as shown in Figure 12 (a). In the example of Figure 12 (a), we can observe that HERALD scheduled all the layers in model A first and those in model B afterward. In contrast, breadth-first ordering interleaves all the layers from each model, as an example in Figure 12 (b) shows. Because depth-first ordering tends to have long idle time because of the layer dependence within a model, depth-first ordering often results in larger overall latency. However, depth-first ordering requires less number of context changes than breadth-first ordering. Those two methods exploit the characteristics of DNN workload that has linear dependence chains within each DNN model and independent among DNN models.

Because layer ordering occurs before the schedule construction, HERALD’s layer ordering engine does not have full information about the exact time considering dependences. Therefore, layer execution order can result in inefficient schedules like Figure 12 (b) where layer 2, 3, and 4 of model B can be scheduled right after the layer 1 of model A. To prevent such inefficient schedules, the layer ordering engine consults the schedule construction engine and perform look-ahead to fix redundant idle time.

Schedule Construction. After layer assignment and ordering steps are done, HERALD constructs a schedule based on the assignment and ordering considering dependence among layers. The schedule construction engine places each layer at the earliest schedulable, which is determined by the worst case latency of the target accelerator and the corresponding model. In Figure 13, we present two examples of schedule construction intermediate steps. For conciseness, we denote each layer in the following format later: layer(modelID,layerNumber). In Figure 13 (a), the scheduler tries to place layer(A,2) (red box) on the accelerator 2. Since the layer(A,1) on the accelerator 1 finishes later than the layer(B,1), the accelerator 2 needs to wait until the accelerator 1 finishes layer(A,1). Therefore, layer(A,2) is scheduled at latency(modelA), not latency(Acc2). In Figure 13 (b), the scheduler tries to place layer(B,2) (blue box) on the accelerator 1. In contrast to the case in Figure 13 (a), layer(B,2) has no stall because of dependence since layer(B,1) finishes earlier than layer(A,1) on the accelerator 1. Therefore, layer(B,2) is scheduled on latency(Acc1), not latency(modelB). Repeating the same method, HERALD constructs a schedule with the minimum latency that follows the given layer assignment and execution order.
Figure 11. An overview of layer scheduling in HERALD. Circled numbers represent a layer in each model.

Table 3. Two heterogeneous DNN workloads used for the evaluation. We combine models discussed in Table 2 to generate those two heterogeneous DNN workloads.

| Workload | Model              | # of instances |
|----------|--------------------|----------------|
| A        | ResNet50           | 2              |
|          | Unet               | 4              |
|          | MobileNetV2        | 4              |
| B        | ResNet50           | 2              |
|          | Unet               | 2              |
|          | MobileNetV2        | 4              |
|          | BR-Q Handpose      | 2              |
|          | Focal Length DepthNet | 2          |

Memory Constraints. When HERALD schedules layers on an HDA, the scheduler checks if a candidate schedule requires more memory than the entire accelerator has estimated based on the execution model we discussed in Section 4.1 If the scheduler identifies such memory size violations among scheduled layers in an intermediate schedule, the scheduler defers the execution of layers scheduled later that caused the memory size violation.

5 Evaluation

We evaluate HDA designs and layer execution schedules generated by HERALD over two edge device workloads.
Table 5. HDA design points with the best EDP for each experiment setting. The experiment setting refers to the workload and accelerator ID we list in Table 3 and Table 4. In BW and PE partitioning, the pair indicates the amount of resources for NVDA and Shi-diannaor style sub-accelerators, respectively.

| Setting   | BW partitioning | PE Partitioning |
|-----------|-----------------|-----------------|
| A, Small Acc. | 40 / 24         | 48 / 16         |
| A, Large Acc. | 224 / 32        | 1536 / 2560     |
| B, Small Acc. | 48 / 16         | 1536 / 2560     |
| B, Large Acc. | 128 / 128       | 12032 / 4352    |

We estimated latency and energy of HDAs in Figure 14, where it contains two sub-accelerator styles; Shi-diannaor and NVDA styles. Overall, we observe that many heterogeneous designs out-perform the interpolated designs between two monolithic designs we combine, which is marked as red dotted lines in Figure 14 (a) - (d). On average, compared to the best monolithic design with the lowest EDP, the best heterogeneous design provided 20.0% EDP improvements.

Impact of HW resource partitioning. In Table 5, we list the hardware resource partitioning results of design points with the best EDP for each experiment configuration. As we observe in Table 5, the partitioning results are not trivial, which motivates an automated tool like HERALD.

Impact of workloads. Each row of Figure 14 shows the latency-energy space of two monolithic designs and heterogeneous design candidates HERALD explored on two different workloads we discussed in Table 3. Figure 16 (a) summarizes the impact of the workloads, which shows the average latency, energy, and EDP improvements of the best EDP heterogeneous accelerator compared to the best monolithic accelerator for each experiment. We observe that HDAs provided higher benefits in workload B than in workload A. This is because workload B includes more DNN models as shown in Table 3, which provides more heterogeneity to exploit and parallelization opportunities over two sub-accelerators.

Impact of accelerator size. Each column of Figure 14 shows the design space of small and large accelerator we list in Table 4. Overall, the design points of the large accelerator were more efficient than those of the small accelerator. In workload A, 56.8% of heterogeneous designs upon the small accelerator had smaller EDP than two baseline monolithic accelerator styles and their interpolated heterogeneous designs (red dotted lines) while 98.2% of heterogeneous designs upon the large accelerator had smaller EDP than those baselines. However, in workload B, heterogeneous designs upon both of the small and large accelerators had more than 93.2% designs more efficient than the best baseline for each. Therefore, both of the workload and accelerator size, or available hardware resources for sub-accelerators, affect the efficiency of resulting HDAs. When the heterogeneity and the number of layers in a workload is small, a large number of hardware resources (number of PEs, NoC bandwidth, and memory size) can improve the efficiency of heterogeneous accelerators.

Impact of scheduling algorithm. We evaluate the EDP of HDAs with a baseline scheduler and HERALD’s scheduler. For the baseline, we implement a scheduler with EDP-greedy layer assignment and depth-first layer ordering discussed in Section 4.3. In Figure 16 (b), we show the average EDP improvements of the best HDA design points for each experiment configuration based on baseline and HERALD’s scheduler. Across all the experiment settings using HERALD’s scheduler, HERALD identified efficient designs with 24.9% average EDP improvements while the baseline scheduler provided 13.6% EDP improvements on average. In addition, on the small accelerator with workload A, the baseline scheduler-based optimized EDP was worse than the best monolithic baseline while HERALD’s scheduler identified an optimized schedule that provides
HERALD: Optimizing Heterogeneous DNN Accelerators for Edge Devices

Figure 14. The design space of HDAs for workloads listed in Table 3

Figure 15. Design space of two- and three-way heterogeneous DNN accelerators on workload A and B

Figure 16. The impact of workload and scheduling on the EDP of HDAs. (a) The average latency, energy, and EDP improvements of HDAs compared to the best monolithic accelerator for each workload. (b) The average EDP improvements compared to the best monolithic design using baseline and HERALD’s scheduler. Improved EDP over baseline.

Impact of the number of sub-accelerators. We present the design space of three sub-accelerators in Figure 15 using Shi-diannao, NVDLA, and Eyeriss style dataflows for large accelerator setting with 16K PEs. Most HDA design points out-performed interpolated design points among monolithic accelerators with the three dataflows. However, in both of the workloads, design points with the best EDP are based on the combination of accelerators with NVDLA and Shi-diannao style dataflows with 3.2% and 6.0% lower EDP than the best three-way HDA on workload A and B, respectively. This suggests that simply deploying many accelerators is not always beneficial, and HDA designers need to select dataflows that can complement each other carefully.

6 RELATED WORK

DNN Dataflows and Accelerators. Shi-diannao (Du et al., 2015) is a CNN accelerator designed to be embedded near sensors, which exploits convolutional reuse via an output-stationary style dataflow. Eyeriss (Chen et al., 2016b) 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 (Alwani et al., 2016) exploited fine-grained pipelined layer parallelism that minimizes activation data movement among memory hierarchy. Flexflow (Lu et al., 2017) is a DNN accelerator that supports three distinct dataflow styles on a DNN accelerator with an analytic model used to identify the best dataflow based on PE utilization. Tensor Processing Unit(TPU) (Jouppi et al., 2017) is a systolic array-based DNN accelerator designed for cloud workload data centers, which includes 256×256 PEs and 28MiB on-chip memories. MAERI (Kwon et al., 2018b) is a flexible dataflow DNN accelerator that also efficiently supports irregular dataflows resulting from sparsity, cross-layer mapping (Alwani et al., 2016), and so on. Tangram (Gao et al., 2019) is a DNN accelerator explored pipelined layer parallelism within a model with optimized dataflow for such back-to-back layer execution.

Heterogeneous Accelerators. Chandramoorthy et al. (Chandramoorthy et al., 2015) 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, focus-
ing on general image kernels, not DNNs. Master of None Acceleration (Lottarini et al., 2019) explored a single-chip heterogeneous accelerator for analytical query application, which revealed that the design space of heterogeneous accelerators for the target domain has both beneficial and disadvantageous design points.

7 Conclusion

In this paper, we explored the latency and energy optimization opportunities of heterogeneous DNN accelerators on complex edge device workloads. Because the efficiency of a DNN accelerator depends on dataflow, workload, and hardware design parameters (number of PEs, memory size, memory/NoC bandwidth, etc.), identifying the best heterogeneous DNN accelerator design point with an optimized schedule is challenging. Therefore, we developed HERALD, an automated design space exploration and layer scheduler framework for heterogeneous DNN accelerators. In our case studies, HERALD identified optimized design points and layer schedules, providing 24.93% EDP benefits compared to the best monolithic design we compare. HERALD has presented that the most efficient design point has non-trivial hardware resource partitioning and a naive scheduler can result in EDP degradation, motivating the necessity of HERALD.

References

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

Alwani, M., Chen, H., Ferdman, M., and Milder, P. Fused-layer cnn accelerators. In The 49th Annual IEEE/ACM International Symposium on Microarchitecture, pp. 22. IEEE Press, 2016.

Apple. The future is here - iphone x (neural engine). https://www.apple.com/newsroom/2017/09/the-future-is-here-iphone-x/, 2017.

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

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

Chen, Y.-H., Krishna, T., Emer, J. S., and Sze, V. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. IEEE Journal of Solid-State Circuits, 52(1):127–138, 2016b.

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

Gao, M., Yang, X., Pu, J., Horowitz, M., and Kozyrakis, C. Tanger: 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, pp. 807–820. ACM, 2019.

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

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

He, L., Wang, G., and Hu, Z. Learning depth from single images with deep neural network embedding focal length. IEEE Transactions on Image Processing, 27(9):4676–4689, 2018.

Huawei. HiAI. https://developer.huawei.com/consumer/en/devservice/doc/2020315, 2018.

Huawei. HiAI. https://consumer.huawei.com/en/campaign/kirin-990-series/, 2019.

Ignatov, A., Timofte, R., Chou, W., Wang, K., Wu, M., Hartley, T., and Van Gool, L. Ai benchmark: Running deep neural networks on android smartphones. In Proceedings of the European Conference on Computer Vision (ECCV), pp. 0–0, 2018.

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

Kwon, H., Pellauer, M., and Krishna, T. An analytic model for cost-benefit analysis of dataflows in dnn accelerators. arXiv preprint arXiv:1805.02566, 2018a.

Kwon, H., Samajdar, A., and Krishna, T. Maeri: Enabling flexible dataflow mapping over dnn accelerators via re-configurable interconnects. In International Conference
HERALD: Optimizing Heterogeneous DNN Accelerators for Edge Devices

Li, D., Wang, X., and Kong, D. Deeprebirth: Accelerating deep neural network execution on mobile devices. In *Thirty-Second AAAI Conference on Artificial Intelligence*, pp. 461–475, 2018.

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

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

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

Mark Sandler, Andrew Howard, M. Z. A. Z. and Chen, L.-C. MobileNetV2: Inverted Residuals and Linear Bottlenecks. *arXiv preprint arXiv:1801.04381*, 2019.

NVIDIA. Nvda deep learning accelerator. [http://nvdla.org](http://nvdla.org), 2017.

Park, S.-J., Son, H., Cho, S., Hong, K.-S., and Lee, S. Srfeat: Single image super-resolution with feature discrimination. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 439–455, 2018.

Qualcomm. Snpe. [https://developer.qualcomm.com/docs/snpe/overview.html](https://developer.qualcomm.com/docs/snpe/overview.html), 2018.

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

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

Saining Xie, Ross Girshick, P. D. Z. T. and He, K. Aggregated Residual Transformations for Deep Neural Networks. *arXiv preprint arXiv:1611.05431*, 2017.

Taigman, Y., Yang, M., Ranzato, M., and Wolf, L. Deepface: Closing the gap to human-level performance in face verification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1701–1708, 2014.