Serving Multi-DNN Workloads on FPGAs: A Coordinated Architecture, Scheduling, and Mapping Perspective

Shulin Zeng, *Student Member, IEEE*, Guohao Dai, *Member, IEEE*, Niansong Zhang, Xinhao Yang, Haoyu Zhang, Zhenhua Zhu, *Student Member, IEEE*, Huazhong Yang, *Fellow, IEEE*, and Yu Wang, *Fellow, IEEE*

Abstract—Deep Neural Network (DNN) INFerence-as-a-Service (INFaaS) is the dominating workload in current data centers, for which FPGAs become promising hardware platforms because of their high flexibility and energy efficiency. The dynamic and multi-tenancy nature of INFaaS requires careful design in three aspects: multi-tenant architecture, multi-DNN scheduling, and multi-core mapping. These three factors are critical to the system latency and energy efficiency but are also challenging to optimize since they are tightly coupled and correlated. This paper proposes **H3M**, an automatic Design Space Exploration (DSE) framework to jointly optimize the *architecture*, *scheduling*, and *mapping* for serving INFaaS on cloud FPGAs. H3M explores: (1) the architecture design space with ***H**eterogeneous* spatial ***M**ulti-tenant* sub-accelerators, (2) layer-wise scheduling for ***H**eterogeneous **M**ulti-DNN* workloads, and (3) single-layer mapping to the ***H**omogeneous **M**ulti-core* architecture. H3M beats state-of-the-art multi-tenant DNN accelerators, Planaria and Herald, by up to 7.5× and 3.6× in Energy-Delay-Product (EDP) reduction on the ASIC platform. On the Xilinx U200 and U280 FPGA platforms, H3M offers 2.1-5.7× and 1.8-9.0× EDP reduction over Herald.

Index Terms—Multi-tenancy, deep neural network, multi-core, accelerator, FPGA

1 Introduction

Cloud-backed INFerence-as-a-Service (INFaaS) [1] currently dominates Artificial Intelligence (AI) workloads in data centers. As computing demands of INFaaS continue to grow, data center designers tend to build increasingly larger monolithic DNN accelerators with the multi-node scaled-out capability, such as Google TPUv3 [2] and Microsoft Brainwave [3]. However, simply increasing the number of nodes to accommodate the computing demands is neither scalable nor cost-effective. Recently, there is a clear trend toward enabling multi-tenancy on the single-node accelerator. First, running more DNN services on a single server (or node) reduces communication costs and improves throughput, which benefits the satisfiability of Service Level Agreement (SLA) [4]. Second, cloud service DNN applications usually involve concurrent execution of heterogeneous DNN models [5], such as AR/VR [6] workloads, natural language processing (NLP) [7], and recommendation system [8]. FPGAs offer fine-grained parallelism and reconfiguration flexibility, making them a promising hardware platform for serving dynamic and heterogeneous multi-DNN workloads for INFaaS. To take full advantage of FPGAs, we need to carefully co-design the *architecture*, *scheduling*, and *mapping* of multi-tenant DNN accelerators.

As listed in Table 1, recent studies at the *architecture* level focus on temporal sharing of a single *monolithic* accelerator [4], [9], spatial sharing of *homogeneous multi-core accelerators* (HMCAs) [10], [11], and *heterogeneous dataflow accelerators* (HDAs) [5]. An HMCA consists of multiple identical cores that can communicate with each other via Networks-on-Chip (NoC). Each core is able to run a DNN model independently or share the same workload with other cores. Therefore, HMCA provides the dynamic fission flexibility that can quickly adapt to dynamic load changes [10], While HDA offers a unique optimization dimension, which stems from the diverse preferences of different layers for dataflow [5] (e.g., weight-stationary (NVDLA) [12], output-stationary (ShiDianNao) [13], and row-stationary (Eyeriss) [14]).

This paper explores a novel spatial multi-tenant architecture that incorporates the benefits of both HDAs and HMCAs. On the one hand, the dataflow flexibility of HDAs better accommodates the *model* heterogeneity [5]: Fig. 1 shows that the GNMT model [16] achieves the best Energy-Delay-Product (EDP) on NVDLA-style accelerators [12] using weight-stationary dataflow, and the VGG16 model [17] demonstrates the best EDP on ShiDianNao-style accelerators [13] using output-

- *Shulin Zeng, Niansong Zhang, Xinhao Yang, Haoyu Zhang, Zhenhua Zhu, Huazhong Yang, and Yu Wang are with the Department of Electrical Engineering, Tsinghua University, Beijing 100190, China. E-mail: {zengsl18, yxh21, hy-zhang22, zhuzhenh18}@mails.tsinghua.edu.cn, niansong.zhang@outlook.com, {yanghz, yu-wang}@mail.tsinghua.edu.cn.*
- *Guohao Dai is with the Qing Yuan Research Institute, Shanghai Jiao Tong University, Shanghai, China. E-mail: daiguohao1992@gmail.com.*

Manuscript received 18 January 2022; revised 24 August 2022; accepted 11 September 2022. Date of publication 12 October 2022; date of current version 7 April 2023.
This work was supported in part by the National Natural Science Foundation of China under Grants U19B2019, U21B2031, 61832007, and 62104128, in part by Tsinghua EE Xilinx AI Research Fund, and in part by the Beijing National Research Center for Information Science and Technology (BNRist). (Corresponding authors: Guohao Dai and Yu Wang.)
Recommended for acceptance by P. Milder.
This article has supplementary downloadable material available at https://doi.org/10.1109/TC.2022.3214113, provided by the authors.
Digital Object Identifier no. 10.1109/TC.2022.3214113

TABLE 1
The Comparison of Cloud-Backed DNN Accelerators in Terms of Architecture, Scheduler, and Design Space

Cloud-Backed DNN Accelerator	Homogeneous Multi-Core	Heterogeneous Dataflow	Multi-Tenancy Support	Multi-DNN Scheduler	Bandwidth Scheduler	Design Space Co-Exploration
TPUv3 [2]	✓	X	NA	NA	Fixed	NA
CloudDNN [15]	X	X	NA	NA	Fixed	Arch. & Mapping
AI-MT [9]	X	X	Temporal	heuristics	Fixed	NA
PREMA [4]	X	X	Temporal	heuristics	Fixed	NA
Zeng et al [10]	✓	X	Spatial	NA	Fixed	NA
Planaria [11]	✓	X	Spatial	heuristics	Fixed	NA
Herald [5]	X	✓	Spatial	heuristics	Fixed	Arch. & Scheduling
H3M (This Work)	✓	✓	Spatial	Optimization	Dynamic	Arch. & Scheduling & Mapping

stationary dataflow. On the other hand, The dynamic nature of INFaaS determines that the Query-Per-Second (QPS) and SLA requirements are different for each task (tenant) and change over time. HMCAs allow for running multiple DNN inference tasks concurrently, each allocated with an appropriate number of cores to meet its demand. Such *task*-level dynamic reconfigurability improves resource utilization and reduces costs while meeting the QPS and SLA requirements.

We also need to carefully consider the core-level granularity of HMCAs since there is a trade-off among flexibility, resources, and energy. For example, Fig. 2 illustrates that the logical resources and power consumption of Xilinx Deep learning Processing Units (DPUs) cores [18], which are specially optimized for Xilinx FPGAs. It shows that DPU cores with different computing capabilities do not maintain a strictly linear relationship with the number of PEs (i.e., DSPs). Besides, an 8-core HMCA (B512) consumes nearly 4× more logic resources and 3× more power than a monolithic Xilinx DPU (B4096) under the same number of PEs on the FPGA platform [19]. It is because each small core in the HMCA requires some additional hardware components (e.g., instruction and data controller, switches, and wires).

At the *scheduling* level, multi-DNN schedule decides the execution order and resource allocation for layers from different DNN models. Recent multi-DNN schedulers (listed in Table 1) use heuristics-based algorithms, such as Shortest-Job-First (SJF) and greedy-based methods [4], [5], [11], which heavily rely on pre-designed stationary architectures. Such heuristic-based methods lead to sub-optimal schedule and cannot fit the rapidly evolving hardware platforms (especially FPGAs) in the cloud. Besides, prior studies assume a fixed bandwidth (BW) allocation at *runtime*, which is because the bandwidth allocation of their multi-tenant architectures cannot dynamically adjust after *design* time. Neglecting the optimization space for dynamic bandwidth allocation at *runtime* will lead to wasteful and over-competition for bandwidth resources, thus deteriorating the efficiency and performance [20], [21].

At the *mapping* level, we need to decide how a DNN layer is mapped to the HDAs (i.e., dataflow mapping in terms of loop reordering and loop tiling [22]) or HMCAs (i.e., multi-core mapping of batch, activation, weight, or partial sum parallelization [23]). Since HDAs already contain the mapping information of the specific dataflow (that we optimize at the *architecture* level), we focus on the selection of the multi-core parallelization scheme for HMCAs. Prior work co-explore the *architecture* and *mapping* of monolithic DNN accelerators for single-DNN workloads on FPGAs, since they are highly correlated with each other [15], [24]. However, the introduction of multi-DNN scheduler makes the correlation between *architecture*, *scheduling* and *mapping* more complex, which has not been explored yet. A tuple of the perfectly matched three will improve the resource utilization and maximize energy efficiency.

In this paper, we propose **H3M**, an *architecture*, *scheduling* and *mapping* co-Design Space Exploration (DSE) framework, which fully exploits (1) the architecutre design space with ***H****eterogeneous* spatial ***M****ulti-tenant* sub-accelerators, (2) layer-wise scheduling for a given ***H****eterogeneous* ***M****ulti-DNN* workload on the spatial multi-tenant accelerator, and (3) single-layer mapping onto the ***H****omogeneous* ***M****ulti-core* architecture.

In Section 3, we formulate design space exploration as a constrained optimization problem, and present a three-level encoding for hardware architecture, scheduling, and mapping. We solve the optimization problem with Covariance-Matrix Adaptation Evolution Strategy (CMA-ES) [25]. The workflow is discussed in Section 4. Section 5 presents an optimized HDA implementation on FPGA with dynamic

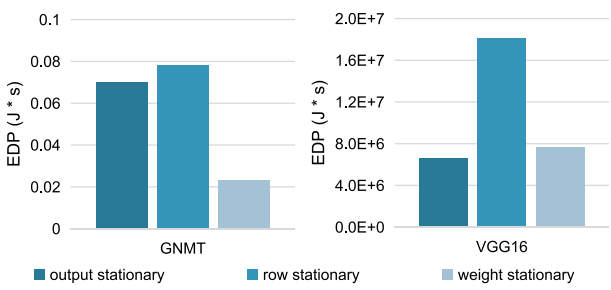

Fig. 1. Comparison of EDP among the three dataflow-style architectures on the GNMT and VGG16 model.

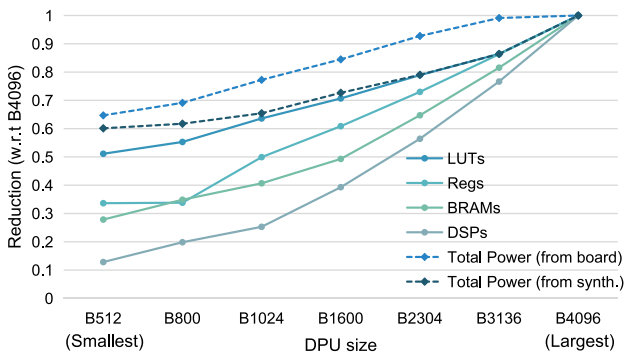

Fig. 2. Resource and power consumption of the eight Xilinx DPUs with different computing abilities [19].

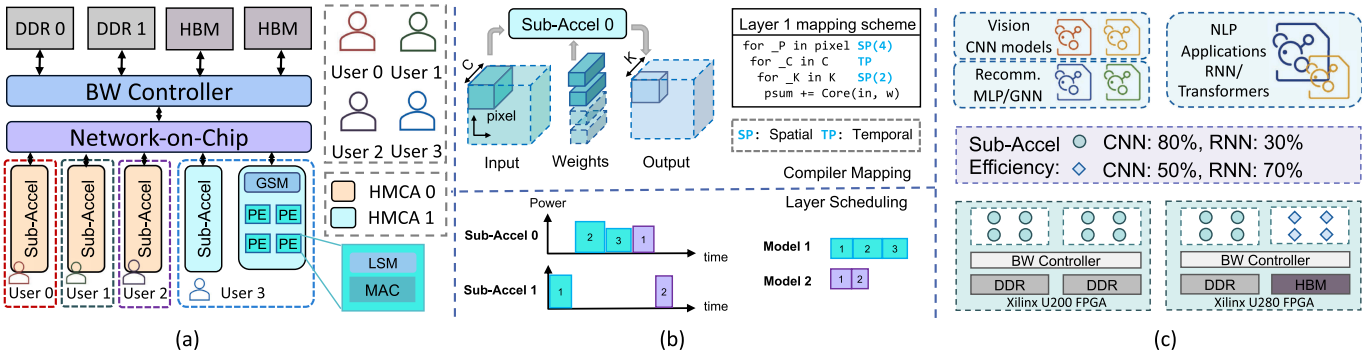

Fig. 3. (a) Spatial Multi-tenant hardware architecture. (b) Multi-DNN compiler mapping and scheduler. (c) Heterogeneous workloads and FPGA systems.

bandwidth control and isolation for security and performance. We empirically evaluate the jointly optimized system on both ASIC and FPGA platforms in Section 6.

The contributions of this paper are:

- We propose a novel spatial multi-tenant architecture that employs the heterogeneous dataflow and homogeneous multi-core architecture at the same time.
- We formulate the multi-DNN scheduling as an optimization problem with dynamic bandwidth allocation considered, by introducing HyperConnect [21] into spatial multi-tenant DNN accelerators.
- We propose an architecture, scheduling, and mapping co-exploration framework, **H3M**, which has a novel encoding format and leverages the CMA-ES algorithm on improving sample efficiency.
- Extensive experiments show that H3M rivals Planaria and Herald by 3.0-7.5× and 1.8-3.6× EDP reduction on the ASIC platform. H3M offers 2.1-5.7× EDP improvement over Herald on the Xilinx U200 FPGA, and 1.8-9.0× on the U280 FPGA.

2 Background and Related Work

2.1 Spatial Multi-Tenant Architecture

As shown in Fig. 3a, a HDA consists of multiple HMCAs with different dataflow and parallelism. We refer each core inside HMCAs as a sub-accelerator. Each sub-accelerator in the spatial multi-tenant architecture is a monolithic DNN accelerator, which is comprised of a Global Scratchpad Memory (GSM) and an array of Processing Elements (PEs) that interconnect with each other in a specific dataflow manner. Each PE contains a Multiply-and-Accumulate (MAC) unit for computation and a Local Scratchpad Memory (LSM) to store activations, weights, and partial sums. For the computation over a tile of a DNN layer, each sub-accelerator first fetches the activations and weights from the off-chip memory (DRAM or HBM) to the GSM in a ping-pong manner, then distributes the data over the PEs for the computation, and finally writes the results back to the GSM.

All sub-accelerators in the spatial multi-tenant accelerator share the off-chip BW through an interconnection module. It is worth pointing out that current commercialized interconnection modules (e.g., AXI interconnect IP on Xilinx FPGAs) lack mechanisms for reserving a fixed portion of off-chip BW to a single sub-accelerator at *design time* and also do not support dynamic reconfiguration at *runtime*. For computing a DNN layer, if the off-chip BW is insufficient for the ping-pong buffer to hide the data loading latency, the computation pipeline will stall, leading to inefficiency.

As listed in Table 1, there are several studies aiming at multi-tenancy. AI-MT [9] optimized the systolic array by considering different resource-usage features inside layers with scheduling methods. Zeng et al. [10] proposed a multi-core DNN architecture to achieve performance isolation, together with a low-overhead runtime reconfiguration compiler. Planaria [11] proposed a systolic array based architecture that could dynamically fission into multiple small pods. Herald [5] employed heterogeneous dataflow architectures for serving multi-DNN workloads.

2.2 Multi-DNN Schedule and Mapping

Multi-DNN schedule decides the execution order and resource allocation of multiple DNN models on multi-tenant accelerators, where a layer is the basic scheduling instance (we refer to each layer as a job in this paper). The job execution order is determined based on priorities for the temporal sharing of a monolithic accelerator [4]. Fig. 3b bottom shows a two-model layer schedule on two sub-accelerators. For spatial multi-tenant accelerators, it is also necessary to consider the resources allocation for each job (e.g., specific HMCA and the number of sub-accelerators in the HMCA). For instance, PREMA [4] proposed a multi-DNN scheduling algorithm for the temporal multi-tenant and preemptive DNN accelerator. Planaria [11] developed an SLA-oriented scheduling algorithm for the spatial multi-tenant DNN accelerator.

When a job is assigned to multiple sub-accelerators, different mapping schemes of multi-core parallelization (e.g., the parallelism among the batch, activation, weight, and partial sum) provide another optimization dimension which is orthogonal to heterogeneous dataflow [11], [23]. For example, Tetris [26] employed a hybrid partition scheme for multi-core accelerators with 3D-stacked DRAM. Tangram [23] co-explored the intra-layer parallelization and inter-layer pipeline for 2D multi-core accelerators. NN-Baton [27] proposed a spatial partition and temporal loop transformation scheme for chiplet-based accelerators. Once the multi-core parallelism scheme is decided, a DNN layer will be partitioned into multiple sub-layers, each of which will run on a sub-accelerator. As shown in Fig. 3b top, a layer is tiled in input dimension by four, and in weight dimension by two. Thus, the layer is spatially mapped to eight sub-accelerators, and temporally

mapped along the channel dimension. Mapping each sub-layer onto the sub-accelerator has been well studied by conventional DNN compilers [24], [28], where the loop order and loop tiling sizes are optimized based on the dataflow of the sub-accelerator [22], which is not the focus of this paper.

2.3 Heterogeneous DNN Workloads and Systems

INFaaS workloads in the cloud mainly include three types of application scenarios: image/video recognition, Natural Language Processing (NLP), and recommendation system. The vision applications are dominated by CNN models [17], [29], which contain many convolution layers (CONV) as the backbone and several fully-connected layers (FC) at the end. The NLP and recommendation systems are dominated by RNN [16], [30] and transformer [31] models, where Multi-Layer Perceptron (MLP) and attention layers are the major components. The diverse operations and shapes of CNN, RNN, and transformer models construct heterogeneous multi-DNN INFaaS workloads.

FPGAs are showing great potential for serving INFaaS workloads in data centers due to their dynamic programmability feature, enabling the spatial multi-tenant architecture to evolve with the dynamic and heterogeneous multi-DNN workloads. Moreover, FPGA systems deployed in the cloud also have heterogeneous characteristics, i.e., different FPGAs have diverse hardware resource constraints (e.g., LUTs, BRAMs, and DSPs) and various main memories (e.g., DRAM and HBM), as shown in Fig. 3c. In this paper, we validate H3M on heterogeneous FPGA systems.

3 Problem Formulation

3.1 Notations

Multi-DNN Workloads. $A = \{a_1, a_2, ..., a_I\}$ is a set of I DNN inference applications from multiple tenants, where each DNN inference application is a 3-tuple $a_i = \langle DNN_i, QPS_i(t), SLA_i\rangle$, $1 \le i \le I$. DNN_i is the target DNN model (e.g., ResNet50), $QPS_i(t)$ is the query load (i.e., the batch size) which changes dynamically over time (e.g., 100fps at t_1 and 50fps at t_2), and SLA_i is the latency constraint from the tenant (e.g., 200ms for each inference).

Heterogeneous and Homogeneous Sub-Accelerators. $H = \{h_1, h_2, ..., h_N\}$ is a set of N HMCAs with a total of M dataflow styles. We denote dataflow styles as $D = \{d_1, d_2, ..., d_M\}$, where $N \ge M$. Each sub-accelerator is defined as a 3-tuple: $h_n = \langle d_n, Core_n, PE_n\rangle$, where $d_n \in D$ denotes the dataflow style, $Core_n$ is the number of sub-accelerators, and PE_n is the number of PEs per sub-accelerator.

Heterogeneous FPGA Systems. $F = \{f_1, f_2, ..., f_S\}$ is a set of S FPGA chips in the serving systems. Each FPGA chip is a 3-tuple $f_s = \langle ResTotal_s, BW_s, freq_s\rangle$, where $ResTotal$ denotes the total hardware resources of LUTs (logic), BRAMs (on-chip memory), and DSPs (hard-core MACs). BW_s represents the available off-chip memory bandwidth, and $freq_s$ denotes the running frequency. Under the resource constraints of a specific FPGA f_s, a spatial multi-tenant DNN accelerator (i.e., a set of N HMCAs) is defined as: $H = \{(d_n, Core_n, PE_n) | \sum Res(d_n, Core_n, PE_n) \le ResTotal_s, 1 \le n \le N\}$.

Multi-DNN Scheduling. Multi-DNN scheduling on the spatial multi-tenant accelerator consists of two major components. First, scheduling function $f_{sche} : f_{sche}(job, t) = \langle 0, 1\rangle$, which decides whether the job executes or not at time t. The scheduling function f_{sche} takes a job queue as input and outputs the execution order among multiple jobs. Second, allocation function $f_{allo} : f_{allo}(job, t) = \langle \delta, Core\rangle$, where $\delta \in H$ denotes the assignment of heterogeneous sub-accelerators and $Core$ represents the allocated number of cores. Optimizing the multi-DNN scheduling with dynamic workloads is a NP-hard problem [32]. Therefore, many previous resource scheduling studies [4], [11] transform the dynamic-workload scheduling into a static scheduling problem, where the central controller (e.g., CPU) packs incoming tasks (i.e., DNN models) over a period of time into batches of jobs (i.e., DNN layers), and a static scheduler processes the batches in an First-Come-First-Serve (FCFS) manner.

Multi-Core Mapping. To explore the mapping of each job, we focus on three types of multi-core parallelization schemes [23]: pixel parallelism along the width or height dimension of activations (denoted as PP), weight parallelism along the output feature maps (denoted as OCP), and partial sum parallelism along the input channel dimension of activations and weights (denoted as ICP). We refer to $PP \times ICP \times OCP$ as the number of cores for spatial mapping the DNN layer over the pixel, weight, and partial sum parallelism, as shown in Fig. 3b.

3.2 Optimization Objectives and Constraints

We choose the widely used optimization objective, *Energy-Delay Product* (EDP), to find the optimal multi-tenant architecture, scheduling, and mapping for a given multi-DNN workload on the target platform, as formulated below:

$$EDP = \left(\sum_{1 \le n \le N} E_{h_n}\right)^{\alpha} \cdot (Latency_{makespan})^{\beta}, \tag{1}$$

where α and β are the weighting factors to control the trade-off between latency and energy consumption. And E_{h_n} is the energy consumption of heterogeneous sub-accelerator h_n running all the jobs assigned to it. As for inference latency, we use the *makespan latency*, i.e., the duration from the beginning of the first job to the end of the last job, as the metric to evaluate the performance of a schedule candidate for a batch of jobs, as shown below:

$$Latency_{makespan} = max(T_{h_1}, T_{h_2}, ..., T_{h_N}), \tag{2}$$

where T_{h_n} is the runtime latency of heterogeneous sub-accelerator h_n to run all the assigned jobs. The objective is to achieve the optimal workload balancing among all the sub-accelerators while minimizing the tail latency (i.e., 95th percentile of the completion time, and makespan latency is the strictest case with 100th percentile).

There are two constraints on the architecture, scheduling, and mapping co-exploration problem. First, the hardware configuration candidate H of the spatial multi-tenant accelerator cannot exceed the resource constraints of the target FPGA platform f_s, as described below:

$$\sum_{1 \le n \le N} Res(d_n, Core_n, PE_n) \le ResTotal_s \tag{3}$$

Second, multiple sub-accelerators sharing the off-chip BW cannot exceed the system BW. If the total BW required by the

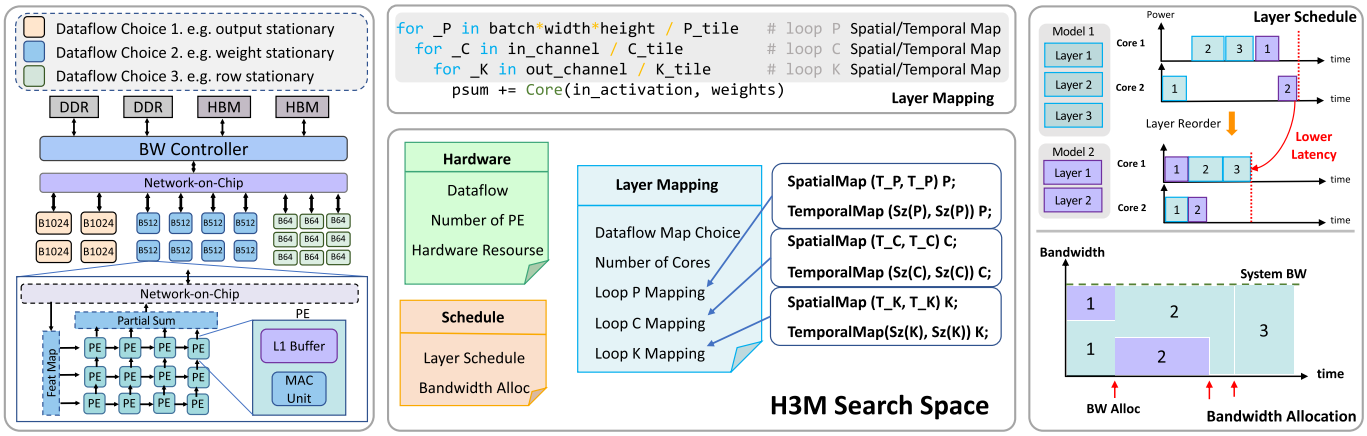

Fig. 4. H3M search space covers architecture, schedule, and mapping. (1) Architecture design space includes sub-accelerator parallelism and dataflow; (2) schedule design space includes layer schedule and bandwidth allocation; (3) mapping design space includes dataflow choice, spatial/temporal map of feature pixel (P), channel (C), and kernel weight (K) dimensions.

concurrent jobs exceeds the BW constraint, they will compete for the off-chip BW resources and interfere with each other. Thus, the total BW usage should not exceed the constraint BW_s of the target FPGA f_s at any time t, as shown below:

$$\sum_{1 \le n \le N} BW^t_{h_n} \le BW_s. \tag{4}$$

3.3 Search Space

As shown in Fig. 4, the search space consists of:(1) the *architecture* design space of the accelerators (H) when deployed to heterogeneous FPGA systems (F) under the given multi-DNN workload (A); (2) the *schedule* design space of the execution order with scheduling function (f_{sche}) and the off-chip bandwidth allocation for parallel jobs; (3) the *mapping* design space of sub-accelerator selection with allocation function (f_{allo}) and choosing the proper combination of spatial/temporal mapping for input feature pixel (P), input channel (C), and kernel weights (K) dimensions.

We use an illustrative example to demonstrate the size of the search space. Table 4 shows the resource constraints of Xilinx U280 FPGAs. Based on the resource utilization of the smallest Xilinx DPU accelerator B512 [18], we can calculate the number of DPU cores to be at most 46. Assuming we have 4 kinds of DPU cores with different dataflow styles, the hardware candidates are at least $C^4_{9024} \cdot (46)^4 = 10^{20}$ within the number of DSPs (i.e., PEs) and DPU cores. We assume that the maximum batch size of arriving jobs is 100, and the possible scheduling combinations are $(100)! = 10^{157}$. For each layer mapping onto the HMCA, the possible combinations of multi-core parallelism parameters are $(46)^3 = 10^4$. Combining the three design spaces, H3M's search space size is $10^{20+157+4\cdot100} = O(10^{577})$, while there are only 10^{18} hardware candidates in Herald's search space. We employ the bio-inspired genetic evolutionary algorithms to find the optimal design point with high sample efficiency [25], [33] in the massive search space.

4 H3M Co-Exploration Framework

4.1 Framework Overview

Fig. 5 shows the optimization loop of the H3M framework. We discuss the optimization process in steps.

Sample Search Space. The optimizer generates samples from the search space. Each sample is a set of three encoding vectors representing a full solution: hardware parameters, mapping scheme, layer schedule, and bandwidth allocation.

Decode. The decoder translates the encoding vectors to solutions. Specifically, it first decodes the hardware encoding vector to get sub-accelerator dataflow choices, core numbers, and PE numbers. Then, based on the decoded hardware information, the decoder translates the mapping encoding vector and the scheduling vector to mapping schemes and priority scores for every layer of all input DNN tasks. A mapping scheme specifies how a layer (job) is executed in parallel on multiple cores: whether it is spatially or temporally mapped in pixel, input channel, and output channel dimensions. It also specifies the spatial mapping factor of each dimension. The priority scores determine the layer schedule on a sub-accelerator and the bandwidth allocation across different cores.

Schedule. From the mapping scheme information, we know the number of cores each layer is assigned to. However, it does not specify *which* cores to use. The jobs are first scheduled in a queue on each HMCA. The scheduler orders the jobs in the queue according to their priority scores and layer dependency. Then, it allocates specific cores to the job in an FCFS manner. For jobs running in parallel, the scheduler allocates their share of off-chip bandwidth according to their normalized priority scores.

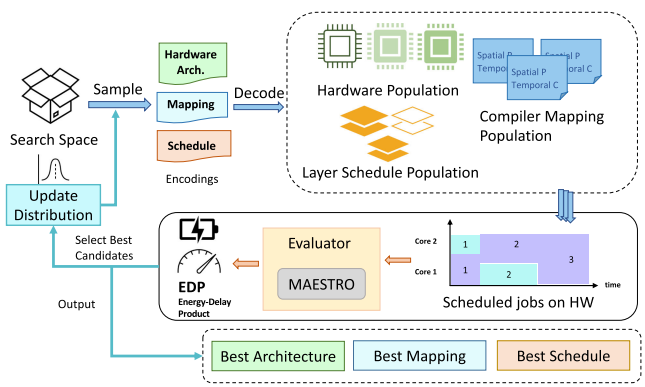

Fig. 5. H3M co-exploration workflow.

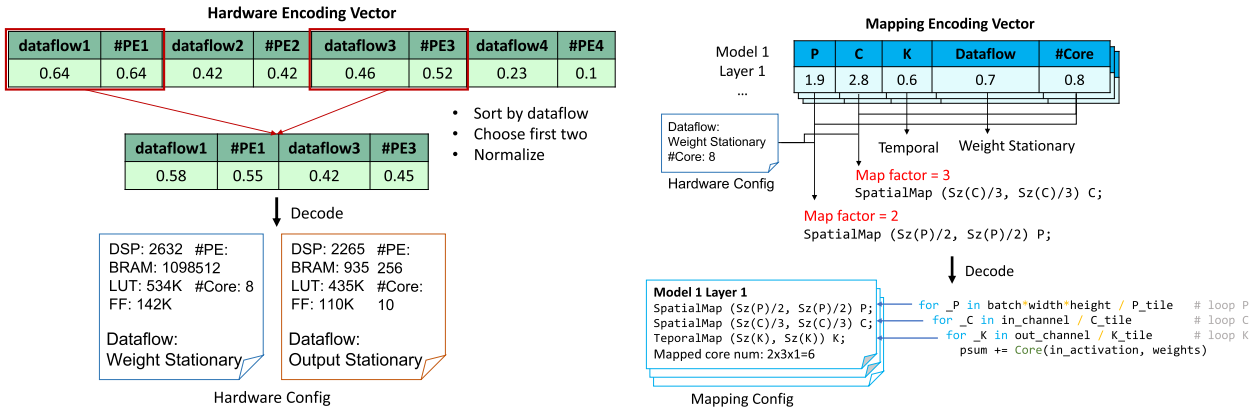

Fig. 6. Hardware and mapping encoding vectors with decode examples. (a) Hardware Encoding Vector. (b) Mapping Encoding Vector.

Simulate. After the schedule and bandwidth allocation are determined, the simulator runs the jobs with the given configuration, and outputs the total latency and energy consumption to calculate EDP.

4.2 Optimization Algorithm

We use CMA-ES [25] to sample the search space and optimize the objective function. CMA-ES is a derivative-free second-order method that estimates a positive-definite covariance matrix, which makes it effective for ill-conditioned and non-smooth problems [33]. CMA-ES is also proven to be reliable and competitive for global optimizations [25]. We choose CMA-ES over heuristic-based methods because of its effectiveness in exploring large design space. The problem formulation and encoding design are general and not limited to this optimization method. CMA-ES models solutions as n-dimensional Gaussian variables. At each evolutionary step, it generates new samples with mean μ and covariance matrix $\mathbf{C}_\sigma$. After evaluation, the parameters are updated with feedback from the objective function.

4.3 The Three-Tier Encoding Format

Hardware Encoding Vector. The hardware encoding vector is an array of $2M$ floating-point numbers, M is the total choices of sub-accelerator dataflow. Each pair of floating-point numbers encodes the hardware resource share and PE number for one choice of sub-accelerator dataflow.

Mapping Encoding Vector. The mapping encoding vector is an array of $5L$ floating-point numbers, with L being the total number of layers. The five fields for each layer are: P, C, K mapping scores, dataflow choice, and the core number factor. A mapping score higher than one indicates spatial mapping in that dimension, otherwise indicating temporal mapping. The dataflow choice determines which HMCA the layer maps to. The number of the core factor is capped between 0 and 1, and it indicates how many cores in the chosen HMCA the current layer maps to.

Scheduling Encoding Vector. The scheduling encoding vector is an array of L floating-point numbers. Each value in the vector is the priority score of the corresponding layer. Layers with higher priority scores are likely to execute earlier in the queue, and be assigned more bandwidth.

4.4 Decoder

We describe how to decode the encoding vectors in steps.

Decode Sub-Accelerator Hardware Configuration. We assume that a single FPGA chip hosts N HMCAs with N dataflow styles. We first sort the M pairs of values in the hardware encoding vector and take the first N pairs. As the first value in the pair encodes hardware resource share, we normalize the N dataflow score values and divide the hardware resources (e.g., LUTs, FFs, BRAMs, DSPs) according to the normalized values. The second value in the pair encodes PE numbers. Assume there are P predefined PE number choices. We fix $P-1$ thresholds, and assign PE numbers according to which interval the PE number value falls into. Given hardware resources and the number of PE in each core, we calculate the number of cores in each HMCA. Fig. 6a is an example of a hardware encoding vector with four dataflow choices. From the four dataflows, we choose dataflow 1 and 3 because of their higher scores, and normalize the scores to divide up the available hardware resources. Given that PE score 0–0.5 maps to 256 PEs in each core, and 0.5–1 maps to 512 PEs, we decode the PE numbers of cores in each HMCA. Finally, we calculate the core number in each HMCA. After the hardware encoding vector is decoded, we know the two HMCAs have 8×B1024 cores with weight stationary dataflow and 10×B512 cores with output stationary dataflow.

Decode Compiler Mapping. Each layer is associated with five fields in the mapping encoding vector. First, we decode the "core type" field to know which HMCA the layer maps to. For N HMCAs, we fix $N-1$ thresholds, and decide the core type by the interval. Since hardware information is decoded, we know the total core number $Core_n$ of the chosen type. p, c, and k are the spatial mapping scores of input feature pixel (P), input channel (C), and kernel weights (K) dimensions. We first bound the scores by:

$$\begin{aligned} p &= max(p, 1) \\ c &= max(c, 1) \\ k &= max(k, 1). \end{aligned} \tag{5}$$

Dimensions with scores ≥ 1 are spatially mapped, otherwise temporally mapped. We decode the spatial mapping factor for spatially mapped loops by:

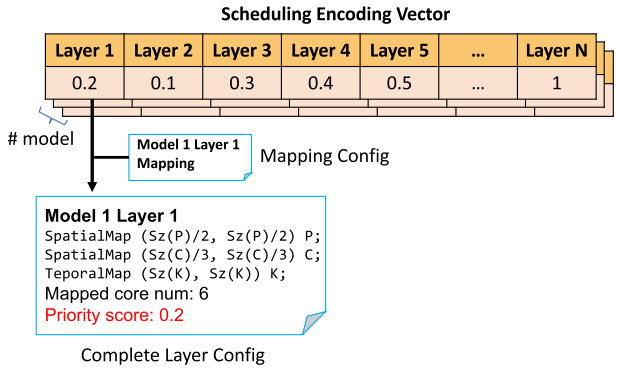

Fig. 7. Scheduling encoding vector.

$$PP = max\left(\left\lfloor usage \times Core_n \times \frac{p}{p \times c \times k}\right\rfloor, 1\right)$$
$$ICP = max\left(\left\lfloor usage \times Core_n \times \frac{c}{p \times c \times k}\right\rfloor, 1\right)$$
$$OCP = max\left(\left\lfloor usage \times Core_n \times \frac{k}{p \times c \times k}\right\rfloor, 1\right), \quad (6)$$

where PP, ICP, and OCP are the spatial mapping factors of feature, input channel, and output channel dimensions, $0 < usage \leq 1$ is the core usage score, denoted as `#Core` in Fig. 6b. The total core number this layer maps to is $PP \times ICP \times OCP$. For the example shown in Fig. 6b, P and C dimensions are spatially mapped with factors 2 and 3, dimension K is temporally mapped, and the total mapped core number is 6. `Sz(P)` is the size function of loop P, therefore `Sz(P)/2` is the tiling size of input feature pixel dimension.

Decode Priority Score. The scheduling vector stores the priority scores for each vector. As shown in Fig. 7, decoding the priority score is straightforward. We simply assign the values to corresponding layers to get the complete configuration. The priority score will be compared and normalized during layer scheduling and bandwidth allocation.

4.5 Evaluator

H3M allocates bandwidth for scheduled jobs, and evaluate the latency and energy consumption with a simulator.

Layer Schedule and Dynamic Bandwidth Allocation. Since layer schedule is part of the optimization, the schedule is known and fixed at runtime. Therefore, H3M does not require a dynamic scheduler as a hardware module or a piece of software in the runtime. However, the off-chip bandwidth is dynamically allocated based on the normalized priority score of parallel jobs. We discuss the implementation of dynamic bandwidth controller in Section 5.2.

Simulator. Each layer's latency and energy consumption is evaluated by MAESTRO [22] simulator. The MAESTRO simulator supports diverse dataflow choices and allows hardware customizations (PE number, scratchpad sizes, NoC latency, bandwidth). The scheduler sets up the hardware resource and bandwidth configuration for the simulator, and calls the simulator to evaluate the latency and energy.

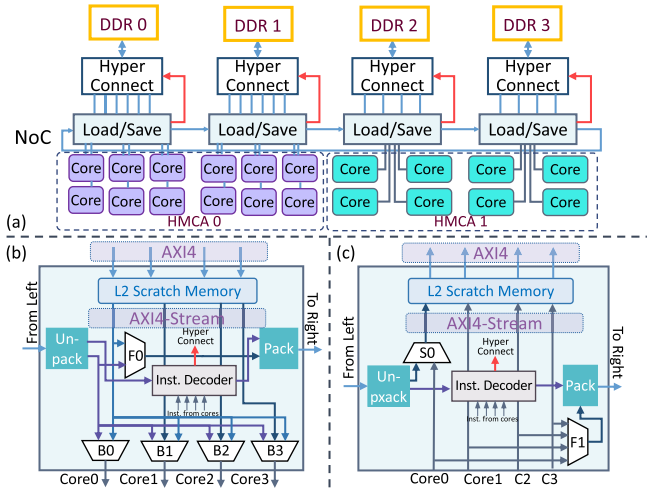

Fig. 8. (a) Top-level architecture of multi-tenant DNN accelerators on the Xilinx FPGA U200 platform. And illustrations of the (b) Load and (c) Save data movement module.

5 Implementation on FPGAs

5.1 Spatial Multi-Tenant Architecture

5.1.1 Top-Level Architecture

Fig. 8a shows the top-level architecture of the spatial multi-tenant accelerator on the Xilinx U200 FPGA having four DDR memory channels. The implementation on Xilinx U280 FPGA is similar, which has two DDR and two HBM. The sub-accelerators and off-chip memory channels are grouped together according to the total number of DDR memory channels and available HBMs. The number of sub-accelerators and dataflow choices within each group is determined by the H3M DSE framework. The inter- and intra-group data movement is managed by the Load/Save data movement modules connected by a Network-on-Chip. The dynamic bandwidth controller in each group manages the data movement between sub-accelerators and off-chip memory channels, while also providing multi-tenant performance isolation and dynamic bandwidth allocation.

5.1.2 Load/Save Data Movement Module

The Load/Save data movement module has forwarding and broadcast control for *Load* instructions and write control for *Save* instructions to the local shared buffer or off-chip DDR memory. Figs. 8b and 8c show the Load and Save data movement module design, the data paths are separated for clarity. It is responsible for: (1) reading data from the left neighbor or off-chip memory to the local cores; (2) forwarding data from the off-chip memory or left neighbor to the right neighbor; (3) writing data from local cores to the local shared buffer or off-chip memory; (4) configuring the dynamic bandwidth controller for runtime bandwidth allocation; and (5) merging the identical read requests into one and broadcast the fetched data. The Load/Save data movement module has the following components:

Network-on-Chip (NoC) forwards the instructions and data between groups. We leverage a directed and light-weight NoC introduced by Hoplite [34]. As shown in Fig. 8b. As the uni-directional NoC takes up much fewer hardware resources than the bi-directional NoC, its width can be designed to

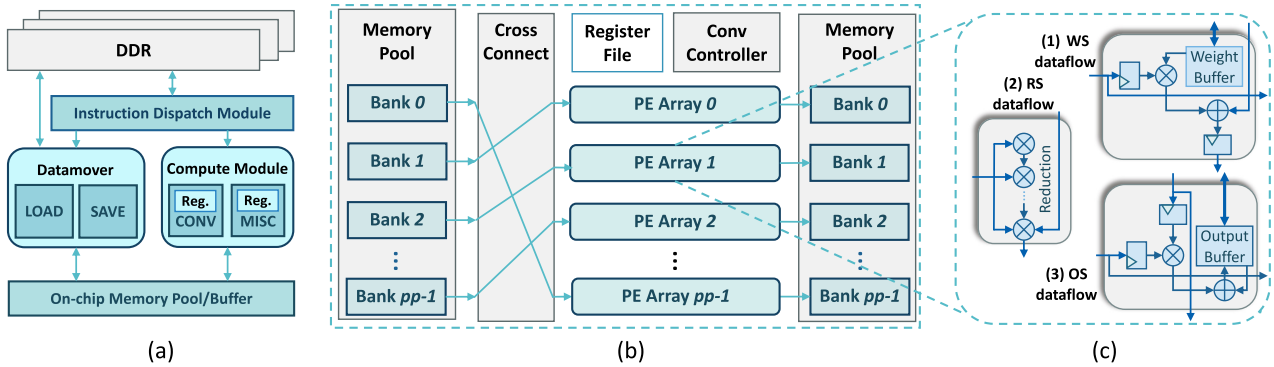

Fig. 9. DNN sub-accelerator architecture based on Xilinx DPU: (a) top level architecture, (b) convolution module architecture, and (c) PE architecture with three different dataflow styles supported .

match the total width of the off-chip memory bandwidth, thus minimizing the performance interference.

Instruction Decoder generates control signals for all of the multiplexers and the dynamic bandwidth controller to manage broadcast, forwarding, and save directions. If identical data is requested, the decoder initiates only one read request to the dynamic bandwidth controller. Meanwhile, it changes the off-chip bandwidth allocation from being distributed equally across multiple cores to having the full bandwidth exclusively to one port for data broadcast. Besides, the instruction decoder configures the dynamic bandwidth controller based on the *Init* instructions of each job, thus enabling runtime bandwidth reconfiguration.

Control Multiplexers are controlled by the Instruction Decoder. In Fig. 8b, multiplexer $B0$ and $F0$ control data forwarding. Multiplexer $B0$ to $B(n-1)$, where n denotes the number of cores in each group, control broadcasting forwarded data or local data. In Fig. 8c, multiplexer $S0$ controls saving unpacked data from the neighbor group to the local shared buffer or DDR. Multiplexer $F1$ selects the intermediate data to be forwarded for multi-core parallelization (e.g., partial sum for input channel parallelism ICP).

*Pack/Unpack Module.*We pack data and instructions into frames for inter-group communication. The unpack module extracts the instruction for decoding, and the pack module concatenates instructions with data to form a frame.

5.1.3 DNN Sub-Accelerator Architecture

In this paper, the basic templates of the DNN sub-accelerator architecture are based on Xilinx DPUs [18]. Since a DPU is a Xilinx proprietary functional block, we implement it from Angel-Eye [35], which is a basic implementation of a Xilinx DPU, and continuously optimize it to keep up with the performance of Xilinx DPUs. As shown in Fig. 9a, the top-level architecture of Xilinx DPUs contains five modules: Local Instruction Decoder and Scheduler (LIDS), data loader module (LOAD), data writer module (SAVE), convolution operator module (CONV), and non-convolution operator module (MISC). The LIDS is responsible for the decoding of instructions and the local scheduling of the other four modules, which correspond to the four instructions.

Fig. 9b illustrates the architecture of CONV module, where there are pp PE arrays, each with a parallelism of $p_{pe} = icp \times ocp$. The computation parallelism of one DPU core (p_{dpu}) can be calculated by:

$$p_{dpu} = 2 \cdot pp \cdot p_{pe} = 2 \cdot spp \cdot icp \cdot ocp \quad (\text{OPs/cycle}), \tag{7}$$

where pp, icp, and ocp represent the pixel, input-channel, and output-channel parallelism. Xilinx DPUs employ the weight-stationary style dataflow for each PE array. More specifically, there are ocp parallel PE channels, each with icp PEs to perform MAC operations in parallel. To support different styles of dataflow, the PE array of DPUs is modified to enable different dataflow styles (e.g., output-stationary and row-stationary dataflow in Fig. 9c). By enabling heterogeneous dataflow on the PE array level, we implement DPU-based accelerators with heterogeneous dataflow and homogeneous multi-core architecture.

5.2 Dynamic Bandwidth Controller Implementation

In this section, we introduce the dynamic bandwidth controller implemented based on AXI HyperConnect [21].

5.2.1 Bandwidth Reservation

Bandwidth reservation for each sub-accelerator is essential to guarantee performance isolation for multi-tenant sharing. For example, a memory-intensive DNN layer with low-priority running on a sub-accelerator can occupy the unlimited BW and inject a lot of delay into the sub-accelerator running a high-priority DNN layer with hard latency constraints. We implement the bandwidth reservation mechanism by *limiting data transactions to a specific threshold number over a periodic time window*, which has been verified in [20]. The threshold for each sub-accelerator can be configured by the hypervisor, thus ensuring the predictable performance of running jobs and avoiding performance interference due to multi-tenant sharing on the off-chip memory bandwidth.

5.2.2 Security Isolation

For a generic FPGA virtualization solution [36] in the cloud, users can either select the pre-designed DNN accelerators provided by cloud vendors, or upload their own designed DNN accelerators into the partial reconfigurable regions of the FPGA. Since these closed-source DNN accelerators are black-box for cloud vendors, it is crucial to ensure security isolation among different DNN accelerators for multi-tenant sharing. Current commercialized AXI interconnection modules (e.g., Xilinx AXI interconnect) use round-robin arbitration to solve the conflicts among multiple accelerators. However, it can lead to serious unfair bandwidth allocation

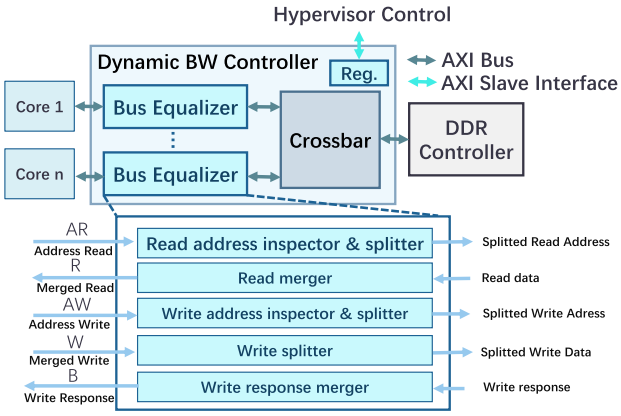

Fig. 10. Dynamic bandwidth controller architecture.

under the case with *heterogeneous burst sizes* [37]. This reveals a serious security issue where a malicious user could bring down the entire FPGA system by uploading the accelerator bitstream with a particularly large burst size as the 'bandwidth stealer'. To tackle this issue, we employ the technique proposed in [37], which works by *equalizing the burst size of each sub-accelerator to a uniform size*. Combining the mechanisms of limiting the number of transactions and the data size in a predictable manner, we can safely ensure both the performance and security isolation on FPGAs.

5.2.3 Runtime Reconfigurability

As discussed in Section 2.1, current commercialized AXI interconnection modules are not able to change their configuration during runtime. We enable the runtime reconfigurability by *exposing an AXI interface for the hypervisor to configure internal registers at runtime*. More specifically, the proposed dynamic bandwidth controller has three major configurable parameters. The first is the threshold h to limit the number of AXI transactions for the bandwidth allocation of each sub-accelerator. The second is the uniform burst size b for fair bandwidth allocation and security isolation. As discovered in [37], a smaller b allows for a fairer bandwidth allocation, but introduces some more latency overhead for the overall transactions, thus an empirical value of b as 16-word is taken to meet the best trade-off. The third is the period T, which has impacts on the total bandwidth utilization rate depending on the specific workload [20]. The hypervisor can dynamically adjust the period T to gradually achieve the optimal bandwidth utilization at runtime.

5.2.4 Architecture Overview

Fig. 10 shows the architecture overview of the dynamic bandwidth controller based on AXI HyperConnect [21], where the bus equalizer is proposed as the key module to incorporate with the conventional AXI interconnection module. For the first functionality, the bus equalizer realizes bandwidth reservation by limiting the number of outstanding transactions based on the threshold h. The bus equalizer has a separate internal counter for each sub-accelerator. Once the number of transactions set by the threshold h is reached, the Central Control Unit (CCU) will suspend further transaction requests of the corresponding sub-accelerator. For the second one, the bus equalizer deals with heterogeneous burst sizes with the help of splitters and mergers. When the burst size is larger than the pre-defined uniform burst size b, the splitter will divide the read and write transaction requests into multiple sub-transactions, and the merger will merge the corresponding responses. For the third functionality, an AXI slave control interface is exposed to the hypervisor as a standard memory-mapped device. The hypervisor can read/write internal registers for configuring parameters and monitoring stats at runtime.

TABLE 2
Multi-DNN Workloads With Vision, NLP, and Mixed Applications

Workload	DNN Models
Vision	ResNet50 [29], MobileNetV2 [39], GoogleNet [40], VGG16 [17]
NLP	GNMT [16], ncf [41], Transformer (12 layers) [31]
Mixed	ResNet50 [29], MobileNetV2 [39], GNMT [16], Transformer (12 layers) [31]

The listed CNN and RNN models are extracted from MLPerf [38].

6 Evaluations

6.1 Evaluation Setups

Multi-DNN Workloads. We consider three different workloads: vision, NLP, and a mix of both. The DNN models consist of CNN, RNN, and transformer models, mainly extracted from the multi-stream inference workload of MLPerf [38]. As listed in Table 2, we construct each workload with four different DNN models. For the Transformer model [31], we limit the number of layers to be 12 (i.e., two self-attention layers) for a more balanced workload. Besides, we set the batch size to 1 for each DNN model.

Evaluated Platforms. We evaluate H3M on both ASIC and FPGA platforms. The ASIC platform experiments aim at providing comparative results with state-of-the-art spatial multi-tenant accelerators, i.e., Planaria [11] and Herald [5]. We follow the settings in Herald for fair comparison, as shown in Table 3. On the other hand, FPGA platforms offer a realistic and deployable environment for evaluating the proposed H3M framework on heterogeneous systems. Table 4 shows the available hardware resources and off-chip memory bandwidth of Xilinx Alveo U200 and U280 FPGAs. We employ the API remoting based virtualization method for the PCIe-based FPGA system [42].

Architecture Settings. As summarized in Tables 3 and 4, we set the local memory to 512 KB for each sub-accelerator

TABLE 3
Settings for ASIC-Based Cloud Use Scenarios

PEs	16384
NoC BW	256 GB/s
Off-chip BW	256 GB/s
Global Memory	16 MB
Local Memory	512 KB
Frequency	1 GHz
Accelerator Parallelism	Xilinx B512/800/1024/B1152 and B1600/B2304/B3136/B4096 DPUs
Dataflow	Output/Weight/Row-stationary

TABLE 4
Resources and Settings for Xilinx FPGA Platforms

FPGA Type	Alveo U200	Alveo U280
LUTs	1182K	1304K
FFs	1777K	2607K
DSPs	6840	9024
BRAMs	1602	2016
URAMs	800	960
Off-chip Memory	4 DDRs (64 GB)	2 DDRs, 1 HBM2 (40 GB)
Off-chip BW	77 GB/s	38+460=498 GB/s
NoC BW	77 GB/s	498 GB/s
Accelerator Parallelism	Xilinx B512/800/1024/B1152 and B1600/B2304/B3136/B4096 DPUs	
Local Memory	512 KB	
Global Memory	16 MB	
Frequency	300 MHz	
Dataflow	Output/Weight/Row-stationary	

Each BRAM block is 36 Kb, and each URAM block is 288 Kb.

and the global memory to 16 MB for a fair comparison. We set the NoC BW to be consistent with the off-chip BW as discussed in Section 5.1.1. We assume a running frequency of 1 GHz on the ASIC platform and implement sub-accelerators on the FPGA platform with a frequency of 300 MHz. As discussed in Section 4, we search the heterogeneous dataflow styles, the homogeneous multi-core number, and the computation parallelism (#PE) for each sub-accelerator. Specifically, we select three distinct dataflow styles for evaluation (i.e., weight-stationary (NVDLA) [12], output-stationary (ShiDianNao) [13], and row-stationary (Eyeriss) [14]). Moreover, we consider the Xilinx DPUs with eight different computation parallelisms from 512 to 4096 [18]. For the architecture baselines, we choose Planaria [11] as the homogeneous multi-core baseline, and Herald [5] as the heterogeneous dataflow baseline.

Scheduling Settings. As discussed in Section 3.1, we optimize the scheduling problem in terms of execution order and dynamic bandwidth allocation for a batch of jobs, which are composed of DNN layers from different DNN models. In the evaluation, the number of batched jobs is equal to the total number of DNN layers in the multi-DNN workload (i.e., vision, NLP, and mixed) as presented above. For the scheduling baselines, we implement two heuristic scheduling strategies, i.e., FCFS and SJF, which are widely employed and have been verified to be effective [4], [11]. Both FCFS and SJF are greedy scheduler and do not support dynamic bandwidth allocation at runtime.

Mapping Settings. We search the three multi-core parallelism schemes, i.e., pixel, weight, and partial sum parallelism (*PP*, *ICP*, and *OCP*), for spatial mapping the DNN layer onto the spatial multi-tenant accelerator, as discussed in Section 3.1. Also, we search the resource allocation in terms of sub-accelerator selection and core assignment inside the mapping encoding vector, as discussed in Section 4. For the mapping baselines, we consider two widely used heterogeneous resource allocation methods, i.e., Minimum Execution Time (MET) and Opportunistic Load Balancing (OLB), with a fixed multi-core parallelism scheme. Both MET and OLB are greedy resource allocator, where MET assigns the job to the fastest sub-accelerators, while OLB assigns the job to the most available sub-accelerators.

DSE Settings. As discussed in Section 4.2, we employ CMA-ES [25] as the optimizer to explore the proposed three-level design space. We implement the H3M framework on the top of the *PYCMA* package [43] using Python. We set the number of population for each iteration to be 20, and we find 500 iterations are enough for H3M to find the optimal solutions, which means a total of 10K sampling design points to be evaluated. We run the experiments on the server with two Intel Xeon 4208 CPUs running at 2.1 GHz. We set the α and β as 1 for a balanced EDP objective.

Cost Estimation. As discussed in Section 4.5, we use MAESTRO [22], which is also employed by Herald [5] and reports only 3.9% error compared with RTL simulation, for modeling latency and energy consumption of each sub-accelerator running the target DNN layer on both ASIC and FPGA platforms. As for FPGA platforms, we employ Xilinx Vitis 2021.1 for synthesis and implementation, and the post-synthesis hardware resource utilization reports of all types of modified Xilinx DPUs are used as input to the cost model of H3M. The real-time power consumption of FPGA-based spatial multi-tenant DNN accelerators is collected using on-chip power monitor, which can be accessed by the host CPU through the PCIe interface.

6.2 Evaluation Results

Comparison With Multi-Tenant DNN Accelerators. Fig. 11 shows the comparison results of H3M and the other two multi-tenant DNN accelerator baselines (i.e., Planaria [11] and Herald [5]). The evaluation results over the three workloads show the same trend on the three platforms, where the homogeneous multi-core architecture of Planaria has the worst EDP (except for the vision workload on the Xilinx U280 FPGA), and heterogeneous dataflow architecture of Herald has a better EDP compared with Planaria. Meanwhile, the proposed H3M framework can find the optimal configuration of the multi-tenant DNN accelerator by combining the both worlds of homogeneous multi-core and heterogeneous dataflow architecture, thus achieving the best EDP for all the three workloads on both platforms.

Comparison With Baselines on ASIC. As shown in Fig. 11a, the optimal configurations spotted by the proposed H3M framework on the ASIC platform improve the EDP over the Planaria and Herald by 3.0-7.5× and 1.8-3.6×, respectively. The reasons are two-fold. On the one hand, Herald takes full advantage of different dataflow styles, but it fails to take advantage of the homogeneous multi-core architecture for supporting multi-DNN workloads in a fine-grained manner. The goal of H3M is to have both the model-level dataflow flexibility of HDA and the task-level dynamic fission capabilities of HMCA at the architecture level. On the other hand, the design space for scheduling and mapping also has an important contribution to the EDP improvement. Since both Planaria and Herald use heuristics based mapping and scheduling methods, while H3M co-explores the design space of mapping and scheduling together with architectural parameters. We will further discuss the effectiveness of the proposed scheduling and mapping optimization methodology later.

Comparison With Baselines on FPGA. For evaluation on FPGA platforms, we implement Planaria with the same

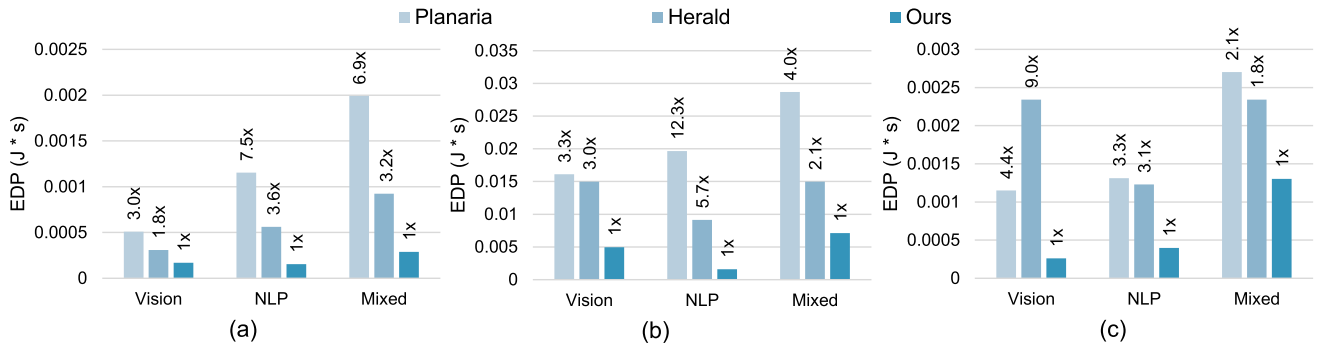

Fig. 11. Comparison results of H3M and other multi-tenant DNN accelerator baselines (i.e., Planaria [11] and Herald [5]) over the three workloads on the different hardware platforms: (a) ASIC, (b) Xilinx U200 FPGA, and (c) Xilinx U280 FPGA.

number of cores as the ASIC platform, which means the architecture of 16-core B2304 and 16-core B4096 DPU sub-accelerators with the same dataflow on the Xilinx U200 and U280 FPGA, respectively. Also, we implement Herald with 5-core B4096 and 8-core B4096 DPU sub-accelerators for the two dataflow styles (i.e., output and weight stationary) on the Xilinx U200 and U280 FPGA, respectively. And we constrain multiple B4096 cores for each dataflow style in Herald to perform only a single job at the same time, to stay consistent with the way Herald is executed.

Fig. 11b illustrates that the EDP improvement of H3M over the Planaria and Herald baselines ranging from 3.3-12.3× and 2.1-5.7× on the Xilinx U200 FPGA, respectively. Similar results of the EDP improvement on the Xilinx U280 FPGA, which are 2.1-4.4× for Planaria and 1.8-9.0× for Herald, can also be observed in Fig. 11b. Compared to the ASIC platform, we can find that the average EDP improvement is greater for the FPGA platform. This is because we need to search PE parameters (i.e., DSPs), but also need to consider other hardware resources such as LUTs and BRAMs on the FPGA platform, resulting in more room for optimization of hardware architecture.

Comparison With Mapping and Scheduling Baselines. Since the mapping and scheduling for multi-DNN workloads are highly coupled with each other, we evaluate both together. As discussed above, we compare the mapping and scheduling optimization results of H3M with several heuristics baselines, including scheduling baselines (i.e., FCFS and SJF) and mapping baselines (i.e., MET and OLB). A complete baseline is the combination of them. For instance, FCFS-MET is a valid strategy, where FCFS is to schedule the execution order of the batched jobs, and MET is to map the job to the available sub-accelerators. Moreover, we fix the hardware configurations to be consistent with Herald to ensure a fair comparison.

Fig. 12 shows the comparison results of H3M and other heuristics multi-DNN mapping and scheduling baselines (i.e., SJF-OLB, FCFS-OLB, SJF-MET, and FCFS-MET) over the three workloads on the ASIC and FPGA platforms. As for the ASIC platform, H3M achieves an average of 1.5×, 3.8×, and 4.6× EDP improvement over the baselines on the vision, NLP, and mixed workload, respectively. We can see the same trend on FPGA platforms, but with better EDP improvements. H3M outperforms the baselines by 1.9-2.9×, 10.3-15.8×, and 12.4-25.7× over the three workloads on the Xilinx U200 and U280 FPGA platform. We can see from Fig. 12 that the heuristics baselines have the worst performance under the mixed workload, where the layers show more heterogeneity. While H3M can find the optimal mapping and scheduling for heterogeneous multi-DNN workloads by collaboratively optimizing on the mapping and scheduling design space for the spatial multi-tenant hardware architecture. What's more, these baselines fail to take advantage of the potential design space for dynamic bandwidth allocation, leading to wasted and competing hardware resources. In contrast, H3M can maximize bandwidth utilization by finding optimal runtime bandwidth allocation schemes through co-exploration.

Optimal Architecture Configuration. We list the optimal architecture configurations spotted by H3M on the Xilinx U200 and U280 FPGA in Tables 5 and 6, respectively. We discover some interesting insights here. First, for the vision

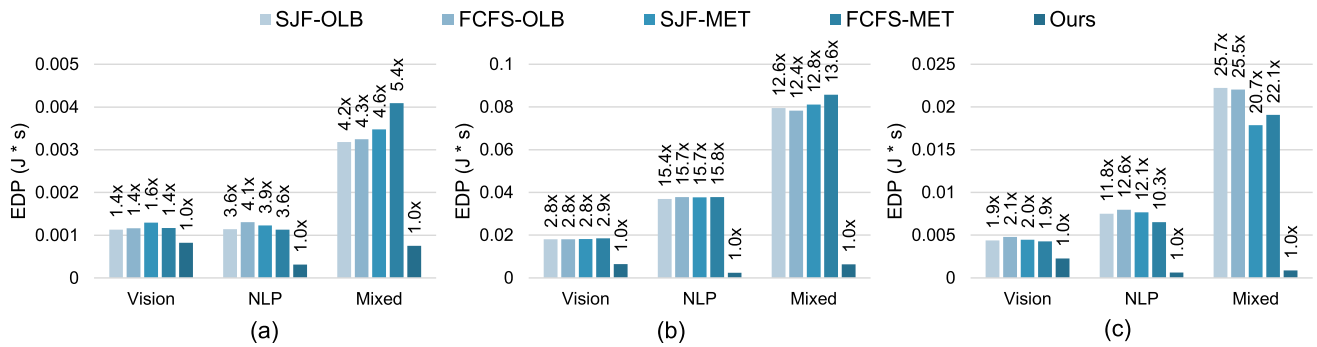

Fig. 12. Comparison results of H3M and other heuristics mapping and scheduling baselines (i.e., FCFS, SJF, MET, and OLB) over the three workloads on the different hardware platforms: (a) ASIC, (b) Xilinx U200, and (c) U280 FPGA.

TABLE 5
Optimal Configuration and Resource Utilization of H3M Over the Three Workloads on the Xilinx U200 FPGA

Modules		Optimal Configs	LUTs	FFs	BRAMs	URAMs	DSPs
HyperConnect		-	12.1K	5.2K	0	0	0
Load/Save Module			21.7K	51.2K	0	0	0
NoC			12.3K	16.5K	0	0	0
Cores	Vision	ws:23*B512 + rs:15*B800	830.4K	1278.4K	968	323	3048
	NLP	os:15*B800 + ws:8*B3136	629.9K	1195.3K	978	326	4682
	Mixed	ws:8*B3136 + rs:4*B4096	452.0K	1088.2K	875	292	5208
Total Utilization		(Vision/NLP/Mixed)	74%/57%/42%	76%/71%/65%	60%/61%/55%	40%/41%/37%	45%/69%/76%

TABLE 6
Optimal Configuration and Resource Utilization of H3M Over the Three Workloads on the Xilinx U280 FPGA

Modules		Optimal Configs	LUTs	FFs	BRAMs	URAMs	DSPs
HyperConnect		-	15.6K	7.5K	0	0	0
Load/Save Module			24.8K	61.4K	0	0	0
NoC			50.1K	65.3K	0	0	0
Cores	Vision	ws:16*B1152 + rs:37*B800	1091.0K	2042.2K	1700	567	6110
	NLP	os:33*B512 + ws:10*B3136	1056.6K	1969.2K	1443	481	6118
	Mixed	os:12*B3136 + ws:8*B4096	760.9K	1831.9K	1479	493	8840
Total Utilization		(Vision/NLP/Mixed)	91%/88%/65%	84%/81%/75%	84%/72%/73%	59%/50%/51%	68%/68%/98%

workload consisting of CNN models, H3M tends to use more small cores (e.g., B512 and B800). Second, for the NLP workload consisting of RNN and transformer models, H3M tends to use a combination of large and small cores (e.g., B800 and b3136). Third, for the mixed workload consisting of CNN, RNN, and transformer models, H3M tends to use a smaller number of large cores (e.g., B3136 and B4096).

For the first case, the reason is that the computational loads of CNN network layers are smaller compared to RNN network layers, so the vision workload will prefer an all-small-core configuration. For the second case, part of the reason is that there is also some heterogeneity in the different RNN models of the NLP workload. For instance, the GNMT network layers are more than 10 times larger than the ncf network layers. For the third case, which has the most heterogeneity, H3M chooses a smaller number of large cores rather than a combination of large and small cores. Such a surprising result could be due to two reasons. On the one hand, the excessive number of small cores leads to higher power consumption, which indicates that H3M is able to find a good trade-off between power and performance to achieve the best EDP. On the other hand, H3M's co-exploration of the three design spaces can exploit optimization spaces that are not available to the heuristic-based multi-DNN scheduling and mapping baselines.

Resource Utilization on FPGAs. Tables 5 and 6 summarizes the resource utilization of H3M over the three workloads on the Xilinx U200 and U280 FPGA, respectively. The AXI HyperConnect controller, Load/Save module, and NoC consume a total of 3.9% and 6.9% logic resources on the Xilinx U200 and U280 FPGA, respectively. To ensure the feasibility of placing and routing on FPGAs, we set the logic resource utilization threshold to 90%. It can be seen from Table 5 that the hardware resources are not fully utilized. This is because the optimization goal of H3M is EDP, and using all the hardware resources achieves the best latency but not the optimal EDP. This also proves again that H3M can achieve the best trade-off between power and latency.

Ablation Study. We analyze how three design spaces affect overall system performance with an ablation study. From the evaluation results of the single-design-space cases shown in Fig. 13, we observe that the scheduling design

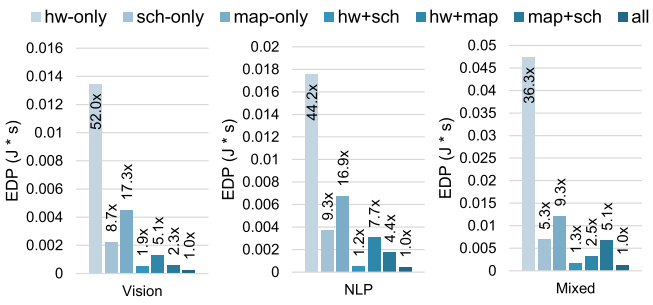

Fig. 13. Ablation study of H3M when searching over the three workloads on the Xilinx U280 FPGA.

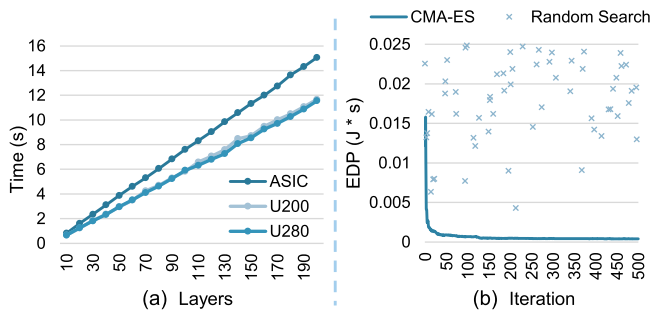

Fig. 14. (a) Sampling runtime of one iteration when searching over the different number of layers of the GNMT model. (b) Sampling efficiency of H3M and random search over the NLP workload on the Xilinx U280 FPGA.

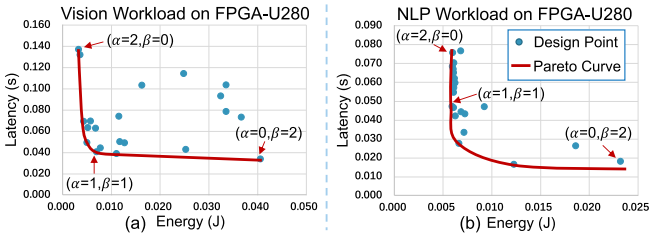

Fig. 15. The Trade-off between latency and energy by adjusting the α and β for the (a) Vision and (b) NLP workload.

space has the greatest impact on the performance (5.3-8.7× difference from the all-design-space case), while the architecture design space has relatively the least impact (36.3-52.0×). As for the cases where we choose two design spaces to construct a combined design space, Fig. 13 illustrates that the combined design space of architecture and scheduling has the greatest impact on the performance with only 1.2-1.9× difference. This further demonstrates that the design spaces of architecture, scheduling, and mapping are highly coupled and correlated with each other, reflecting the need for H3M to optimize all the three collaboratively.

Sampling Efficiency. As discussed in Section 4.3, the length of the hardware and mapping encoding vector is fixed, while the length of the scheduling encoding vector equals to the number of batched jobs (i.e., layers). Fig. 14a shows the sampling runtime of H3M scales linearly with the number of layers for one iteration. When we fix the number of batch jobs to 100, H3M takes 5.8-7.6s for one iteration, and around one hour for a complete exploration with 100 iterations. Fig. 14b demonstrates that H3M can quickly converge to a near-optimal design point in the first 200 iterations, while the EDP mean of random search remains high. It indicates that H3M gradually improves the range of architecture, scheduling, and mapping selection.

Trade-off Between Latency and Energy. We explore different combinations of EDP objectives with (α, β) between $(2, 0)$ (i.e., makespan latency only) and $(0, 2)$ (i.e., energy only) with a step of 0.1. Fig. 15 illustrates that H3M controls the trade-off between the makespan latency and energy, where a larger α and a smaller β result in a better makespan latency at the cost of a higher energy consumption, and vice verse. Moreover, we discover that the more balanced EDP objectives (i.e., (α, β) is close to (1,1)) are more likely to appear on the Pareto frontier. Thus, we set the (α, β) to (1,1) as the default configuration for H3M to ensure a Pareto-optimal and latency-energy balanced result.

7 Conclusion

In this paper, we propose H3M, an architecture, scheduling, and mapping co-exploration framework for FPGAs, which provides the following takeaways. (1) The combination of the *heterogeneous dataflow* and *homogeneous multi-core* architecture offers superior performance over SOTA multi-tenant DNN architectures. (2) Heuristics-based mapping and scheduling algorithms cannot meet the performance demand required by multi-DNN workloads. Instead, we need to formulate it as an *optimization* problem with *dynamic bandwidth allocation* considered. (3) The design spaces of the *architecture*, *scheduling*, and *mapping* are highly coupled and correlated with each other, which demands for a *co-exploration* methodology. (4) H3M rivals the SOTA multi-tenant DNN accelerator baselines, Planaria and Herald, by 3.0-7.5x and 1.8-3.6x EDP reduction on the ASIC platform. H3M offers 2.1-5.7x EDP improvement over Herald on the Xilinx U200 FPGA, and 1.8-9.0x on the U280 FPGA.

References

[1] F. Romero, Q. Li, N. J. Yadwadkar, and C. Kozyrakis, "INFaaS: Managed & model-less inference serving," 2019, *arXiv:1905.13348*.
[2] Google, "Google cloud TPU: Train and run machine learning models faster than ever before," 2021. [Online]. Available: https://cloud.google.com/tpu/
[3] J. Fowers et al., "A configurable cloud-scale DNN processor for real-time AI," in *Proc. IEEE/ACM 45th Annu. Int. Symp. Comput. Archit.*, 2018, pp. 1–14.
[4] Y. Choi and M. Rhu, "PREMA: A predictive multi-task scheduling algorithm for preemptible neural processing units," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit.*, 2020, pp. 220–233.
[5] H. Kwon, L. Lai, M. Pellauer, T. Krishna, Y.-H. Chen, and V. Chandra, "Heterogeneous dataflow accelerators for multi-DNN workloads," in *Proc. IEEE Int. Symp. High-Perform. Comput. Archit.*, 2021, pp. 71–83.
[6] C.-J. Wu et al., "Machine learning at facebook: Understanding inference at the edge," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit.*, 2019, pp. 331–344.
[7] D. Amodei et al., "Deep speech 2: End-to-end speech recognition in english and mandarin," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 173–182.
[8] U. Gupta et al., "DeepRecSys: A system for optimizing end-to-end at-scale neural recommendation inference," in *Proc. IEEE/ACM 47th Annu. Int. Symp. Comput. Archit.*, 2020, pp. 982–995.
[9] E. Baek, D. Kwon, and J. Kim, "A multi-neural network acceleration architecture," in *Proc. IEEE/ACM 47th Annu. Int. Symp. Comput. Archit.*, 2020, pp. 940–953.
[10] S. Zeng et al., "Enabling efficient and flexible FPGA virtualization for deep learning in the cloud," in *Proc. IEEE 28th Annu. Int. Symp. Field-Programmable Custom Comput. Machines*, 2020, pp. 102–110.
[11] S. Ghodrati et al., "Planaria: Dynamic architecture fission for spatial multi-tenant acceleration of deep neural networks," in *Proc. IEEE 53rd Annu. ACM Int. Symp. Microarchit.*, 2020, pp. 681–697.
[12] NVIDIA, "NVIDIA deep learning accelerator," 2017. [Online]. Available: http://nvdla.org
[13] Z. Du et al., "ShiDianNao: Shifting vision processing closer to the sensor," in *Proc. 42nd Annu. Int. Symp. Comput. Archit.*, 2015, pp. 92–104.
[14] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," *ACM SIGARCH Comput. Archit. News*, vol. 44, no. 3, pp. 367–379, 2016.
[15] Y. Chen, J. He, X. Zhang, C. Hao, and D. Chen, "Cloud-DNN: An open framework for mapping DNN models to cloud FPGAs," in *Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays*, 2019, pp. 73–82.
[16] Y. Wu et al., "Google's neural machine translation system: Bridging the gap between human and machine translation," 2016, *arXiv:1609.08144*.
[17] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*.
[18] Xilinx, "Deep learning processor unit (DPU) intellectual property," 2021. [Online]. Available. https://www.xilinx.com/products/intellectual-property/dpu.html
[19] R. Kedia, S. Goel, M. Balakrishnan, K. Paul, and R. Sen, "Design space exploration of FPGA based system with multiple DNN accelerators," *IEEE Embedded Syst. Lett.*, vol. 13, no. 3, pp. 114–117, Sep. 2021.
[20] M. Pagani, E. Rossi, A. Biondi, M. Marinoni, G. Lipari, and G. Buttazzo, "A bandwidth reservation mechanism for AXI-based hardware accelerators on FPGAs," in *Proc. 31st Euromicro Conf. Real-Time Syst.*, 2019, pp. 14:1–14:9.

[21] F. Restuccia, A. Biondi, M. Marinoni, G. Cicero, and G. Buttazzo, "AXI hyperconnect: A predictable, hypervisor-level interconnect for hardware accelerators in FPGA SoC," in *Proc. IEEE 57th ACM Des. Automat. Conf.*, 2020, pp. 1–6.
[22] H. Kwon, P. Chatarasi, V. Sarkar, T. Krishna, M. Pellauer, and A. Parashar, "MAESTRO: A data-centric approach to understand reuse, performance, and hardware cost of DNN mappings," *IEEE Micro*, vol. 40, no. 3, pp. 20–29, May/Jun. 2020.
[23] M. Gao, X. Yang, J. Pu, M. Horowitz, and C. Kozyrakis, "TANGRAM: Optimized coarse-grained dataflow for scalable NN accelerators," in *Proc. 24th Int. Conf. Architectural Support Program. Lang. Oper. Syst.*, 2019, pp. 807–820.
[24] X. Zhang et al., "DNNBuilder: An automated tool for building high-performance DNN hardware accelerators for FPGAs," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, 2018, pp. 1–8.
[25] N. Hansen, "The CMA evolution strategy: A comparing review," in *Towards a New Evolutionary Computation*, Berlin, Germany: Springer, pp. 75–102, 2006.
[26] M. Gao, J. Pu, X. Yang, M. Horowitz, and C. Kozyrakis, "TETRIS: Scalable and efficient neural network acceleration with 3D memory," in *Proc. 22nd Int. Conf. Archit. Support Program. Lang. Oper. Syst.*, 2017, pp. 751–764.
[27] Z. Tan, H. Cai, R. Dong, and K. Ma, "NN-baton: DNN workload orchestration and chiplet granularity exploration for multichip accelerators," in *Proc. IEEE/ACM 48th Annu. Int. Symp. Comput. Archit.*, 2021, pp. 1013–1026.
[28] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-based accelerator design for deep convolutional neural networks," in *Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays*, 2015, pp. 161–170.
[29] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.
[30] H. Sak, A. W. Senior, and F. Beaufays, "Long short-term memory recurrent neural network architectures for large scale acoustic modeling," *Proc. 15th Annu. Conf. Int. Speech Commun. Assoc.*, 2014, pp. 338–342.
[31] A. Vaswani et al., "Attention is all you need," in *Proc. Adv. Neural Informat. Process. Syst.*, 2017, pp. 5998–6008.
[32] B. Mor, D. Shabtay, and L. Yedidsion, "Heuristic algorithms for solving a set of NP-hard single-machine scheduling problems with resource-dependent processing times," *Comput. Ind. Eng.*, vol. 153, 2021, Art. no. 107024.
[33] Y. Lin, M. Yang, and S. Han, "NAAS: Neural accelerator architecture search," 2021, *arXiv:2105.13258*.
[34] N. Kapre and J. Gray, "Hoplite: A deflection-routed directional torus NoC for FPGAs," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 10, no. 2, pp. 1–24, 2017.
[35] K. Guo et al., "Angel-eye: A complete design flow for mapping CNN onto embedded FPGA," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 37, no. 1, pp. 35–47, Jan. 2018.
[36] Y. Zha and J. Li, "Virtualizing FPGAs in the cloud," in *Proc. 25th Int. Conf. Archit. Support Program. Lang. Oper. Syst.*, 2020, pp. 845–858.
[37] F. Restuccia, M. Pagani, A. Biondi, M. Marinoni, and G. Buttazzo, "Is your bus arbiter really fair? Restoring fairness in AXI interconnects for FPGA SoCs," *ACM Trans. Embedded Comput. Syst.*, vol. 18, no. 5, pp. 1–22, 2019.
[38] V. J. Reddi et al., "MLPerf inference benchmark," in *Proc. ACM/IEEE 47th Annu. Int. Symp. Comput. Archit.*, 2020, pp. 446–459.
[39] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobilenetV2: Inverted residuals and linear bottlenecks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 4510–4520.
[40] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 2818–2826.
[41] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua, "Neural collaborative filtering," in *Proc. 26th Int. Conf. World Wide Web*, 2017, pp. 173–182.
[42] S. Zengetal ., "A unified FPGA virtualization framework for general-purpose deep neural networks in the cloud," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 15, no. 3, pp. 1–31, 2021.
[43] Github, "Python implementation of CMA-ES," 2021. [Online]. Available: https://github.com/CMA-ES/pycma

Shulin Zeng (Student Member, IEEE) received the BS degree from the Electronic Engineering Department, Tsinghua University, Beijing, China, in 2018. He is currently working toward the PhD degree with Electronic Engineering Department, Tsinghua University. His research interests include mainly focuses on software-hardware co-design for deep learning, FPGA-based accelerator design, and virtualization in the cloud.

Guohao Dai (Member, IEEE) received the BS and PhD (with honor) degrees from Tsinghua University, Beijing, in 2014 and 2019, respectively. He is joining Shanghai Jiao Tong University, Shanghai, China, as an associate professor. His research mainly focuses on large-scale sparse graph computing, heterogeneous hardware computing, emerging hardware architecture, and etc. He has received Best Paper Award in ASP-DAC 2019, and Best Paper Nomination in DAC 2022 and DATE 2018. He is the winner of the NeurIPS Billion-Scale Approximate Nearest Neighbor Search Challenge, in 2021, the recipient of the Outstanding PhD Dissertation Award of Tsinghua University, in 2019. Currently, he serves as PI/Co-PI for several projects with a personal share of more than RMB 6 million.

Niansong Zhang received the BEng degree in telecommunication engineering from Sun Yat-sen University, in 2020. He is currently working toward the PhD degree with Cornell University. His research interests include heterogeneous compilation, domain-specific language for hardware design, and physical design optimizations for FPGAs. This work is conduced during his research internship with Tsinghua University.

Xinhao Yang received the BS degree from the Electronic Engineering Department, Tsinghua University, Beijing, China, in 2021. He is currently working toward the MS degree in electronic engineering with Tsinghua University, Beijing. His research interests include Neural Network (NN) compiler and heterogeneous compiler.

Haoyu Zhang received the BS degree from the Electronic Engineering Department, Beihang University, Beijing, China, in 2022. She is currently working toward the MS degree in electronic engineering with Tsinghua University, Beijing. Her research interest includes software-hardware co-design for deep learning on FPGA.

Zhenhua Zhu (Student Member, IEEE) received the BS degree from the Electronic Engineering Department, Tsinghua University, Beijing, China, in 2018. He is currently working toward the PhD degree with the Electronic Engineering Department, Tsinghua University. His research interests include mainly focuses on memristor, computer architecture, and processing-in-memory.

Huazhong Yang (Fellow, IEEE) received the BS degree in microelectronics, in 1989, and the MS and PhD degrees in electronic engineering from Tsinghua University, Beijing, in 1993 and 1998, respectively. In 1993, he joined the Department of Electronic Engineering, Tsinghua University, Beijing, where he has been a professor since 1998. He was Awarded the Distinguished Young Researcher by NSFC in 2000, Cheung Kong Scholar by the Chinese Ministry of Education (CME), in 2012, science and Technology Award first prize by China Highway and Transportation Society in 2016, and Technological Invention Award first prize by CME in 2019. His current research interests include wireless sensor networks, data converters, energy-harvesting circuits, nonvolatile processors, and brain inspired computing. He has also served as the chair of Northern China ACM SIGDA Chapter science 2014, general co-chair of ASPDAC'20, navigating committee member of AsianHOST'18, and TPC member for ICCCAS'07, ASQED'09, and ICGCS'10.

Yu Wang (Fellow, IEEE) received the BS and PhD (with honor) degrees from Tsinghua University, Beijing, in 2002 and 2007, respectively. He is currently a tenured professor with the Department of Electronic Engineering, Tsinghua University. His research interests include brain inspired computing, application specific hardware computing, parallel circuit analysis, and power/reliability aware system design methodology. He has authored and co-authored more than 200 papers in refereed journals and conferences. He has received 4 best paper awards from leading conferences, including ASPDAC 2019 and FPGA 2017, and 10 Best Paper Nominations. He is a recipient of DAC under 40 Innovator Award (2018). He served as TPC chair for ICFPT 2019 and 2011, track chair for DATE 2017-2019 and GLSVLSI 2018. Currently, he serves as associate editor of *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* and *ACM Transactions on Design Automation of Electronic Systems (TODAES)*.