Planaria: Dynamic Architecture Fission for Spatial Multi-Tenant Acceleration of Deep Neural Networks

Soroush Ghodrati   Byung Hoon Ahn   Joon Kyung Kim   Sean Kinzer   Brahmendra Reddy Yatham   Navateja Alla

Hardik Sharma*   Mohammad Alian[b]   Eiman Ebrahimi[‡]   Nam Sung Kim[†]   Cliff Young[§]   Hadi Esmaeilzadeh

Alternative **C**omputing **T**echnologies (**ACT**) Lab

University of California, San Diego

*Bigstream Inc.   [b]Kansas University   [†]University of Illinois Urbana-Champaign   [‡]NVIDIA Research   [§]Google Inc.

{soghodra, bhahn, jkkim, skinzer, byatham, nalla}@eng.ucsd.edu   hardik@bigstream.co   alian@ku.edu

eebrahimi@nvidia.com   nskim@illinois.edu   cliffy@google.com   hadi@eng.ucsd.edu

Abstract—**Deep Neural Networks (DNNs) have reinvigorated real-world applications that rely on learning patterns of data and are permeating into different industries and markets. Cloud infrastructure and accelerators that offer INFerence-as-a-Service (INFaaS) have become the enabler of this rather quick and invasive shift in the industry. To that end, mostly accelerator-based INFaaS (Google's TPU [1], NVIDIA T4 [2], Microsoft Brainwave [3], etc.) has become the backbone of many real-life applications. However, as the demand for such services grows, merely scaling-out the number of accelerators is not economically cost-effective. Although multi-tenancy has propelled datacenter scalability, it has not been a primary factor in designing DNN accelerators due to the arms race for higher speed and efficiency. This paper sets out to explore this timely requirement of multi-tenancy through a new dimension: dynamic architecture fission. To that end, we define** Planaria[1] **that can** *dynamically fission (break)* **into multiple smaller yet full-fledged DNN engines at runtime. This microarchitectural capability enables** *spatially co-locating* **multiple DNN inference services on the same hardware, offering simultaneous multi-tenant DNN acceleration. To realize this dynamic reconfigurability, we first devise breakable omni-directional systolic arrays for DNN acceleration that allows omni-directional flow of data. Second, it uses this capability and a unique organization of on-chip memory, interconnection, and compute resources to enable fission in systolic array based DNN accelerators. Architecture fission and its associated flexibility enables an extra degree of freedom for task scheduling, that even allows breaking the accelerator with regard to the server load, DNN topology, and task priority. As such, it can simultaneously co-locate DNNs to enhance utilization, throughput, QoS, and fairness. We compare the proposed design to PREMA [4], a recent effort that offers multi-tenancy by time-multiplexing the DNN accelerator across multiple tasks. We use the same frequency, the same amount of compute and memory resources for both accelerators. The results show significant benefits with (soft, medium, hard) QoS requirements, in throughput ($7.4\times$, $7.2\times$, $12.2\times$), SLA satisfaction rate (45%, 15%, 16%), and fairness ($2.1\times$, $2.3\times$, $1.9\times$).**

Index Terms—**Accelerators; Deep Neural Networks; DNN; DNN Acceleration; Multi-Tenancy; Spatial DNN Task Co-Location; Multi-Tenant DNN Acceleration; Dynamic Architecture Fission; Omni-Directional Systolic Arrays**

---

[1]Planaria is a species which, when an individual is cut (*fissioned*) into pieces, all pieces can regenerate to fully formed individuals.

I. INTRODUCTION

The end of Dennard scaling [5] and diminishing benefits from transistor scaling [6–8] has propelled an era of Domain-Specific Architectures [9]. As such, accelerators are put in the spotlight to enable performance improvements necessary for emerging workloads [10]. Although, most recently, accelerators have made their way into consumer electronics, edge devices, and cell-phones (e.g., Edge TPU [11], NVIDIA Jetson [12], and Apple Bionic Engine [13]), their limited computational capacity still necessitates offloading most of the inference tasks to the cloud. In fact, INFerence-as-a-Service (INFaaS) [14], has become the backbone of the deployed applications in Voice Assistants [15, 16], Smart Speakers [17], and enterprise applications [18–20], etc. Cloud-backed inference currently dominates the market [21–24] and is enabled by various forms of custom accelerators, such as Google's TPU [1], NVIDIA T4 [2], Microsoft Brainwave [3], and Facebook's DeepRecSys [25].

As the demand for INFaaS scales, one solution could be continuously increasing the number of accelerators in the cloud. Although intuitive, this approach is neither cost-effective nor scalable with the ever-increasing demand for DNN services. On the other hand, multi-tenancy, where a single node is shared across multiple requests, has been a primary enabler for the success of cloud-computing in current scale. Without multi-tenancy, it is hard to even fathom the progress and future of data-centers and cloud-based computing. In fact, the broader research community invested more than a decade of efforts to develop solutions across the computing stack to bring forth seamless and scalable multi-tenant cloud execution models [26–48]. Nonetheless, multi-tenancy has not been a primary factor in the design of DNN accelerators because of the arms race to design the fastest accelerator, the utmost recency of accelerator adoption in datacenters, and challenges associated with multi-tenancy in accelerators. The datacenter accelerator designs revealed–for instance in Google's TPU [1] or Microsoft Brainwave [3]– tend to show results focused on running a single neural network model as fast as possible. Even the MLPerf benchmark
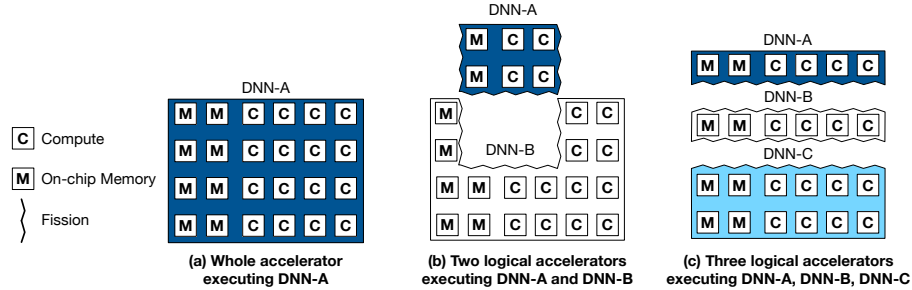
Fig. 1: Illustration of possible fission schemes of Planaria with their corresponding spatially mapped DNNs.

suite [49, 50] keeps this single-model focus for both training and inference. But experience in cloud accelerator systems shows that keeping multiple models simultaneously resident on an accelerator has deployment benefits. Beyond just multiple customers sharing an accelerator, there is demand for multi-tenancy inside of a single application. For example, speech recognition and voice synthesis systems tend to require multiple models in deployment and can significantly benefit from multi-tenancy and co-location [51]. Yet, only this year PREMA [4] has explored a scheduling algorithm that time-multiplexes a DNN accelerator across different DNNs through preemption.

This paper, on the other hand, sets out to explore this timely, yet unexplored dimension of multi-tenancy in the *architecture design* of DNN accelerators. This work presents Planaria, where the key idea is *dynamically fissioning* the DNN accelerator at runtime to *spatially co-locate* multiple DNN inferences on the same hardware. To that end, the paper makes the following contributions:

1) **Dynamic architecture fission for spatial multi-tenant execution.** This paper introduces and explores the dimension of dynamic fission in DNN accelerators. This innovation enables *simultaneous* execution of multiple DNN acceleration threads to be *spatially co-located* on the same hardware substrate. This exclusive runtime reconfigurability in DNN acceleration offers a new degree of freedom in task scheduling to promote utilization and fairness while meeting the Qulity of Service (QoS) constraints.

2) **Microarchitecture design for dynamic fission.** The paper devises a concrete microarchitecture as an instance of dynamic fissionable architectures by delving into the design challenges associated with offering this technology on TPU [1]-like systolic designs. Specifically, we devise omni-directional systolic arrays for DNN acceleration that permits flow of data in all four directions from each elements in the array. This low-cost additional flexibility expands the fission possibilities leading to significant energy reduction and performance gains. To coordinate fission with appropriate on-chip and off-chip data transfer, we arrange these omni-directional systolic arrays in on-chip pods that also comprise specialized interconnection and shared storage for each pod.

3) **Task scheduling for spatial multi-tenant execution.** To leverage architecture-level fission, the paper defines a task scheduling algorithm that breaks up the accelerator with respect to the current server load, DNN topology,

and task priorities, all while considering the latency bounds of the tasks. As the following results indicate, this scheduling algorithm can harness fission capability to simultaneously co-locate DNNs to significantly improve utilization, throughput, QoS, and fairness.

We evaluate Planaria using three INFaaS workload scenarios made up of inference requests to nine diverse DNN benchmarks. Each scenario is evaluated under three different Quality of Service (QoS) requirements. We compare the proposed design to PREMA [4], a recent effort that offers multi-tenancy by time-multiplexing the DNN accelerator across multiple tasks. We use the same frequency, the same amount of compute and memory resource for both accelerators. Our results show that Planaria outperforms PREMA in terms of throughput by $7.4\times$, $7.2\times$, and $12.2\times$ for soft, medium, and hard QoS constraints, respectively. For these set of constraints, Planaria also offers 45%, 15%, and 16% increase in Service-Level Agreement (SLA) satisfaction rate, respectively. At the same time, Planaria improves fairness by $2.1\times$, $2.3\times$, and $1.9\times$.

Our results suggest that exploring simultaneous spatial co-location through architecture fission and balanced task scheduling provides significant benefits. To this end, dynamic architecture fission paves the way for spatial multi-tenancy that can offer a unique direction in the era of cloud-scale acceleration of DNNs.

II. DYNAMIC ARCHITECTURE FISSION: CONCEPTS AND OVERVIEW

The objective is to enable multi-tenant execution of DNNs by spatially co-locating multiple DNN tasks on a single accelerator. To do so, the underlying accelerator needs to dynamically fission at runtime into smaller pieces of *logical full-fledged accelerators* that can execute their pertinent DNN. Figure 1 illustrates three possible examples for the proposed accelerator fission and how the accelerator can spatially execute multiple DNN tasks simultaneously. Generally, a DNN accelerator is a collection of on-chip memory banks M and compute resources, e.g. Multiply-ACcumulate (MAC) units C. Figure 1(a) illustrates that if a DNN task with high priority or tight slack to meet the QoS constraint is dispatched to the accelerator, an entire accelerator is dedicated to the task to expedite its completion. In contrast, Figure 1(b,c) show multiple DNN tasks being dispatched simultaneously. To process them all, the accelerator can fission into multiple logical accelerators, each of which

682

executes a given task as shown. Importantly, fission needs to take place at both compute and memory level, since each logical engine is a standalone independent DNN accelerator. Moreover, the amount of compute and memory resources assigned to each logical accelerator ought to be balanced with the computational demand of the dispatched DNNs to maximize the throughput of the accelerator while meeting the QoS constraints. To that end, bringing forth spatial multi-tenant execution requires devising two major components as follows:

Fission microarchitecture. The first component of this work is *a microarchitecture that can fission dynamically into smaller full-fledged accelerators to execute multiple DNNs simultaneously*. Section III starts from a baseline monolithic DNN accelerator based on systolic array architecture and discusses a set of challenges as well as the design requirements that should be taken into account to fission a monolithic design both at compute and on-chip memory level. Then, Section IV delves into the microarchitectural innards of Planaria, an incarnation of dynamic architecture fission. First, the design of Planaria adds omni-directional data movement in systolic arrays to offer variegated logical fission possibilities. Second, it uses this capability and a unique reorganization of the accelerator, called Fission Pods, to enable fission in systolic array based DNN accelerators. Fission Pods are designed to offer a significant degree of fission flexibility, through specialized connectivity, on-chip memory organization, and omni-directional flow of data in its systolic units. This degree of flexibility is necessary to cope with the varying needs of dispatched DNNs that can be best matched by forming heterogeneous logical accelerators as depicted in Figure 1.

Task scheduler. As the second component of this work, we devise *a task scheduling algorithm that adaptively schedules and assigns the resources to different tasks*. First, the scheduler identifies minimal amount of resources required to execute the DNN while meeting the QoS constraints imposed. Then, it uses a scoring mechanism that congregates task priority and remaining time to distribute the remaining resources on the accelerator to spatially co-locate tasks. This scoring mechanism leads to higher fairness as it considers multiple criteria and flexibility in the accelerator to co-locate multiple DNNs. Importantly, while the spatial co-location improves fairness, the scheduler effectively utilizes the dynamic fission mechanism and considers improving the QoS as its primary design principle. In fact, spatial co-location leads to better utilization of the accelerator resources as more than one task can run at the same time. Section V discusses this scheduling mechanism in detail.

III. ARCHITECTURE DESIGN FOR FISSION: CHALLENGES AND OPPORTUNITIES

This section starts by reviewing a monolithic systolic accelerator, similar to TPU [1]. Then, it provides a series of design requirements to enable spatial multi-tenant execution for DNNs.
Monolithic Systolic Array. Figure 2(a) illustrates a monolithic systolic DNN accelerator[2]. The accelerator consists of a 2D

[2]This section uses a 4×4 systolic array as an example for clarity.

array of Processing Elements (PE) to perform matrix multiplications and convolutions, a unified multi-bank Activation Buffer, a 1D array of Output Buffers, and a SIMD Vector Unit to execute the remaining layers such as pooling, activation, batch normalization, and etc. Input activations are stored on-chip in the unified Activation Buffer–generally implemented as a multi-bank scratchpad, where each bank is shared across PEs within a row. Consequently, at each cycle, an input activation is read from an Activation Buffer's Bank and is reused for all the PEs (MAC units) within the row. *At each cycle, each PE forwards the input activation to the PE to its right (horizontal) and the output partial sum to the PE to its bottom (vertical). In short, this is a waterfall-like uni-directional flow of data as illustrated in Figure 2(b).* Finally, the outputs are fed to the SIMD Vector Unit for further processing. The remainder of the section elaborates on how to fission all the components comprising this monolithic accelerator.

A. Fission for Compute and the Need for New Communication Patterns

(1) The need for flexible and cost-effective fission of *compute* resources. Computational characteristics of DNNs such as data reuse and coarse-grained parallelism vary significantly across different networks or even across different layers of a network [52–55]. The systolic array architectures inherently exploit spatial data reuse for input activations along its rows and partial sums along its columns. *However, a monolithic array design provides only a fixed dimension of this spatial data reuse.* Moreover, as shown for TPU [1], *mapping a convolution or matrix multiplication operation to a big monolithic systolic array can lead to underutilization of compute resources.* As such, some layers naturally perform better if they are tiled to smaller chunks and parallelized across multiple smaller arrays, as that would exploit coarse-grain parallelism and yield better resource utilization.

Figure 3 illustrates multiple examples of possible configurations for decomposition of a 4×4 systolic array, where a 2×2 subarray is used as the granularity for fission. Figure 3(a) shows a fission where the systolic array is broken down horizontally into two subarrays, while Figure 3(b) shows an instance of its
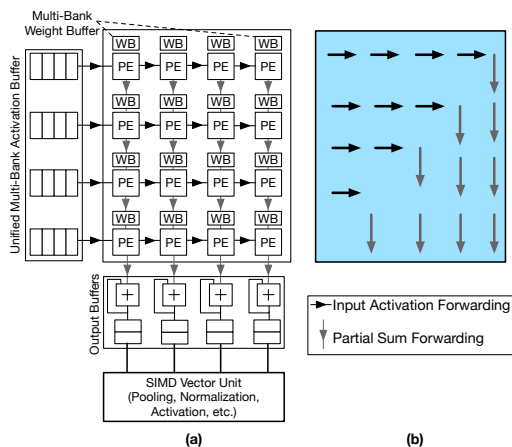

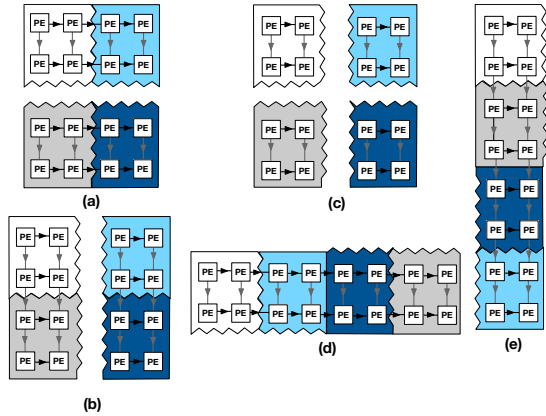**Fig. 2: A monolithic systolic array accelerator.**

Fig. 3: Illustration of possible fission scenarios.

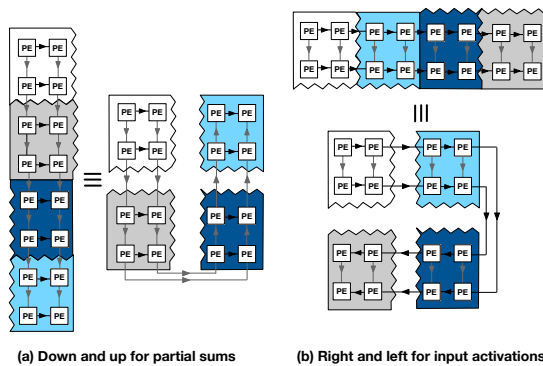**(a) Down and up for partial sums** **(b) Right and left for input activations**

Fig. 4: Omni-directional systolic execution.

vertical fission into two subarrays. Figure 3(c) illustrates another fission in both vertical and horizontal directions, yielding four systolic subarrays. For a layer that requires high coarse grain parallelism, fission in Figure 3(c) would be a good match, while Figure 3(b) would yield the best performance for layers that enjoy more partial sum reuse as well as the coarse-grain parallelism. In another scenario, if a layer requires high input activation reuse, moderate partial sum reuse, and coarse-grain parallelism, fissioning to Figure 3(a) will be the best choice.

Fission granularity. With respect to compute fission, an important design decision is where to break the systolic array. As Figure 2 shows, PEs are connected via two uni-directional links: horizontal and vertical. One extreme option is to replace these links to those that can be dynamically switched on and off to fission the systolic array at the granularity of a single PE. However, such a fine granularity of fission will impose significant overheads. Therefore, *we instead replace a subset of the links to determine the granularity such that they can disconnect a subarray of the PEs instead of a single PE.* The design space exploration of the subarray size is discussed in Section VI-B2.

(2) The need for new and flexible patterns of *communication* for richer fission possibilities. Figure 3(d,e) illustrates two more fission scenarios. If a network layer provides significantly higher opportunity in input activation reuse than partial sum reuse, while not requiring high parallelism, a scheme such as Figure 3(d) is desirable, while fissioning to Figure 3(e) will be a better design for significantly high partial sum reuse. Re-

alizing the last two configurations, however, requires additional design considerations. To forward the input activations along four subarray fragments in Figure 3(d) and partial sums in Figure 3(e), the data needs to flow in all directions: right and left for input activations and up and down for partial sums. Figure 4(a) and Figure 4(b) illustrates how the partial sums and input activations need to flow at all directions to realize the desired scenario. To that end, *we propose omni-directional systolic arrays that can forward the input activations and partial sums in all directions as opposed to conventional systolic arrays that always forward the data in just two directions.*

Communication across the fissioned subarrays. In addition to the omni-directional intra-subarray data movement, there is a need for low-cost inter-subarray communication that also facilitates reconfigurability. As such, *we propose a bi-directional ring bus to connect the fissioned systolic subarrays instead of other forms of connectivity, e.g. crossbar, that would impose significant overheads. The bi-directional nature is to extend omni-directional communication along the subarrays. The links of the ring are configurable in that they can be either off to fission two subarrays or on to forward input activations and partial sums.*

(3) Enabling full-fledged logical accelerators through fission for the SIMD Vector Unit. To create stand-alone accelerators from the fissioned units, the SIMD Vector Unit also needs to be broken into smaller segments and coupled with each systolic subarray. Due to the parallel nature of this unit, we divide the original SIMD Vector Unit to smaller segments proportional to the number of systolic subarrays, and designate a segment to each. When systolic subarrays are vertically stacked (e.g., Figure 3(b,e)), a subset of these SIMD segments are bypassed.

B. Fission for the On-Chip Memory and the Need for Reorganizing the Entire Design

Besides the systolic array, the accelerator also requires fissioning the on-chip memory blocks to allocate commensurate storage to the compute units. Memory disaggregation across the chip is crucial for maximizing on-chip resource utilization. That is because, the on-chip buffers bandwidth to the PE subarrays needs to be kept unchanged to supply enough data to keep the PEs busy. Otherwise, fission would diminish utilization instead of improving it, which was a primary objective of this work.

While decomposing Weight Buffer is straightforward due to its coupling within the PEs, fission for the Activation Buffer and Output Buffer is more challenging.

Weight buffer fission. In systolic arrays, each PE harbors a private Weight Buffer that holds a subset of the network parameters. As such, the total Weight Buffer gets broken down naturally during fission as our strategy does not break the PE.

Activation and output buffer fission. Figure 5 illustrates on-chip memory fission for three of the scenarios shown in Figure 3(b,c,d). Each of the scenarios requires different fission scheme for the Activation Buffer and Output Buffer as well as various patterns of connection between the buffers with the systolic subarray, which are not possible in a monolithic design. In the monolithic case, the Activation Buffer is just connected to the leftmost PEs and Output Buffer to the bottom-most PEs.
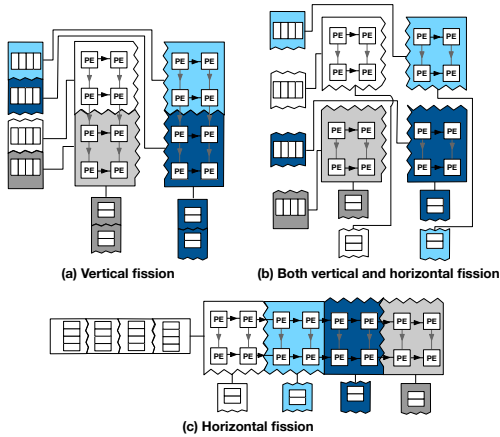
(a) Vertical fission

(b) Both vertical and horizontal fission

(c) Horizontal fission

Fig. 5: On-chip memory fission and connection to subarrays.

However, as Figure 5 depicts, more patterns of connectivity between these buffers and the PEs/subarrays are necessary. To support these variegated patterns, *we reorganize the entire accelerator and devise a microarchitectural block, dubbed Fission Pod, where the Activation Buffer and Output Buffer are co-located in a dedicated memory substrate that is shared amongst a group of connected omni-directional systolic subarrays.* This reorganization and sharing a dedicated memory substrate amongst a *group of subarrays* is crucial to strike a balance between the cost of connectivity in the hardware and achieved utilization of the compute and memory resources.

C. Fission without Reorganization Defeats the Purpose

Figure 6 illustrates a hypothetical case when the systolic array has been partitioned into multiple independent subarrays without properly reorganizing the memory modules. As shown, only the subarray at the left-bottom corner could be utilized, as it would be the only one connected to the Activation Buffer and Output Buffer banks. The other subarrays could not be utilized and would remain idle as illustrated in Figure 6. This underutilization would be a common case if fission happens at granularities other than a single subarray.

Another extreme is illustrated in Figure 7 illustrates where an alternative hypothetical design point connects all the Activation Buffer and Output Buffer banks to all the subarrays. As depicted, this design would require *two high-radix $n \times n$ crossbars*, where n is the number of subarrays. This significantly costly solution is necessary to provide the connectivity patterns discussed in Figure 5 and avoid underutilization of the subarrays. This design point is also not acceptable due to the high-radix crossbars, and can seriously curtail scaling up the compute resources.

Our Fission Pod which reorganizes the subarrays and on-chip memory amortizes this significant overhead, while providing the connectivity patterns required to achieve high utilization of the computer resources as discussed in the next section.

IV. MICROARCHITECTURE FOR FISSION

This section delves into the microarchitecture design of dynamic architecture fission for spatial multi-tenant execution.
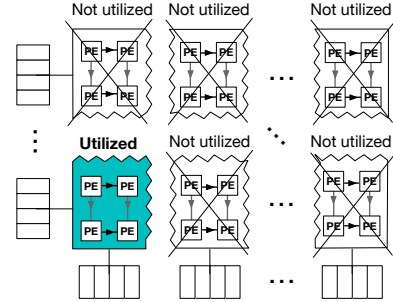

Fig. 6: Underutilization of the subarrays while they are connected to on-chip memory similar to conventional systolic arrays without reorganization of the design. The teal-colored subarray is the only one that can be utilized.
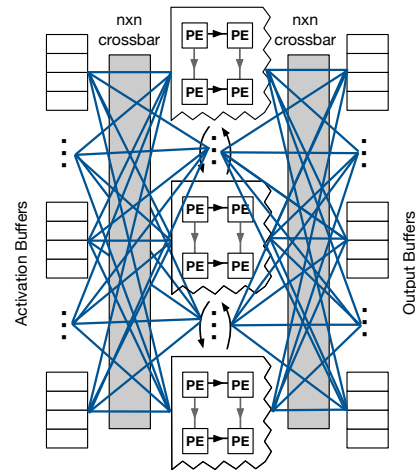

Fig. 7: On-chip memory to subarrays connectivity through high-radix crossbars in an alternative hypothetical design point.

A. Omni-Directional Systolic Array Design

Our novel insight is that, for fission, there is a need for omni-directional pattern of communication in the systolic array to enable richer fission and rearrangement possibilities. An opportunity exists to support this pattern through addition of a low-cost logic to each PE. Figure 8 illustrates how a set of of additional multiplexers around a PE can enable this omni-directional movement. In addition to the normal flow of data (right and down), these multiplexers enable each PE to send input activations to its left and partial sums to the PEs at its top. As highlighted in dark blue, a multiplexer at the left of the PE selects its input from either the activation coming from the right or the left. A de-multiplexer at the right selects which of the left or the right PE should receive the activation. The multiplexer and the de-multiplexer are coupled and are controlled by the same single bit, setting the direction of input activations along the array. Similarly, another pair of multiplexer/de-multiplexer on the north and south of the PE in the Figure 8 control the flow of partial sums. To enable fission and omni-directional inter-subarray data movements, PEs at the boundaries of systolic subarrays are also connected through these multiplexer/de-multiplexer pairs to the corresponding PEs in the adjacent subarrays.

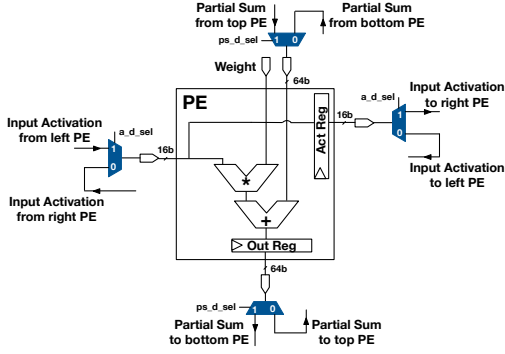**Effect on clock frequency.** Synthesis results show that this

Fig. 8: Switching network for omni-directional systolic array.

extra logic does not reside on the critical path that determines the clock cycle of the systolic subarray. In fact, the critical path is from the Weight Buffer to the Output Register, as access to the on-chip buffer dominates the execution time.

B. Reorganizing the Accelerator Microarchitecture through Fission Pod Design

The key objectives in designing the microarchitecture for dynamic fission are:

1) Creating multiple stand-alone and full-fledged logical accelerators to enable spatial co-location.
2) Enriching the fission possibilities as much as possible to serve various computational needs of co-located DNNs.
3) Maximizing the PE subarray utilization.
4) Maximizing the on-chip buffers utilization and their bandwidths to subarrays.

While meeting these design objectives, the following design constraints need to be considered:

1) Imposing minimal power/area overhead to the hardware
2) Maintaining the baseline clock frequency

With these design objectives and constraints in mind, we propose a microarchitectural unit, called Fission Pod, which *interweaves the on-chip memory with the systolic subarrays and provides balanced cooperation of these components*. Figure 9 illustrates the design. As shown, at the center of this unit, an on-chip memory substrate, called Pod Memory, is placed and connected to a *group* of systolic subarrays. Following discusses the cooperation and communication of the subarrays and the Pod Memory.

Memory-compute interweaving in Fission Pod. A conventional systolic array harbors a unified multi-bank Activation Buffer and a unified multi-bank Output Buffer on their left and bottom, respectively (see Figure 2). When a systolic array is broken into four subarrays as depicted in Figure 9, the aforementioned buffers are moved to Pod Memory and are broken down into four corresponding independent multi-bank buffers. These four buffers are connected to the four systolic subarrays via two 4×4 crossbars to maximize flexibility and fission possibilities that require various patterns of connectivity between on-chip buffer and a group of (size four in this work) systolic subarrays, while also maximizing the PE subarray and on-chip buffer utilization. One crossbar is for reading from Activation Buffers and the other
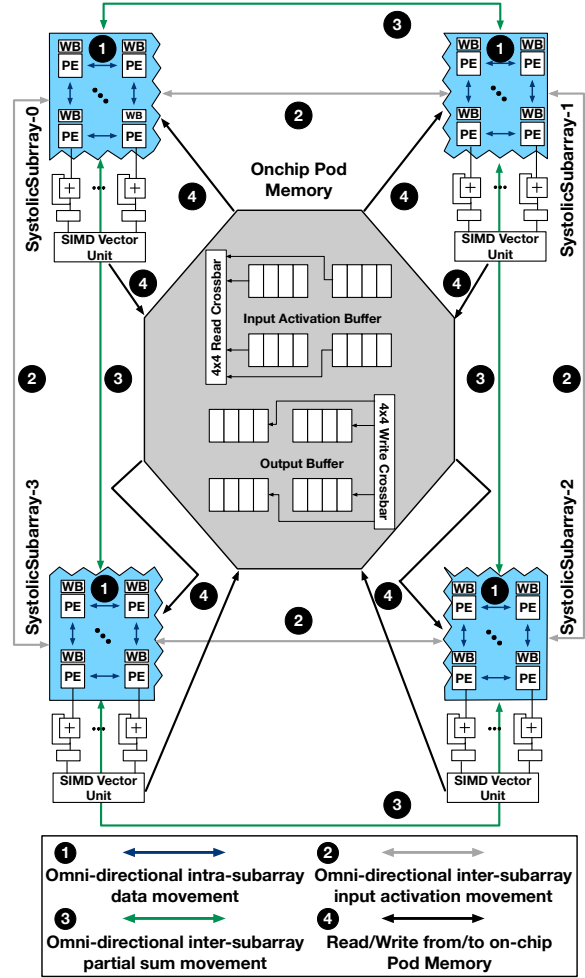

Fig. 9: Fission Pod.

for writing to Output Buffers of the Pod Memory. This design point is in contrast with the design shown in Figure 7, where all the subarrays are connected *globally* to all on-chip buffers through high-radix crossbars, leading to significant power/area overheads, as in here lower-radix crossbars are sufficient due to reorganizing a group of subarrays and on-chip buffers in a Fission Pod (first design constraint).

Intra Fission Pod data communication. The systolic subarrays are also connected to one another via two sets of bi-directional ring buses. One bus is to pass activations between omni-directional subarrays (❷) and the other is to forward the subarray partial sums (❸). These buses enable realizing different fission possibilities while leveraging the omni-directional feature of proposed subarrays. For instance, to realize the fission scheme in Figure 3(d), where the subarrays reconstruct a fat and short array, the activation ring bus will chain the subarrays. The SystolicSubarray-0 in Figure 9 sends the activations to SystolicSubarray-1, and so on and so forth. Since for fission scheme in Figure 3(d), there is no need for partial sum forwarding, the partial sum ring bus will be switched off.

Clock frequency consideration. The two ring buses are pipelined with 12 registers to alleviate any potential critical
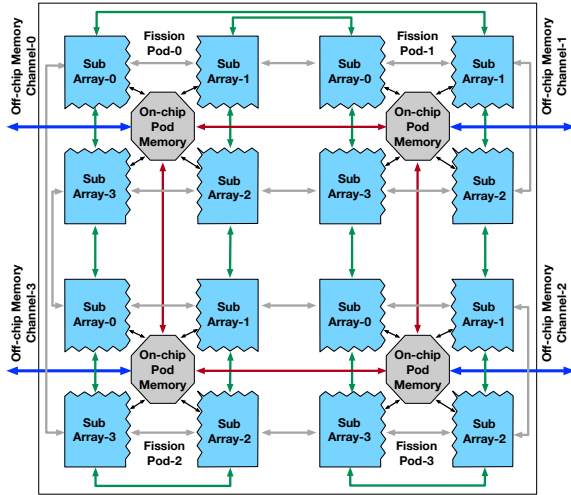
Fig. 10: Overall architecture of Planaria.

paths due to the connectivity between the subarrays. Pipelining is feasible due to natural behavior of systolic arrays that pump wavefronts of data continuously. As such, the added connectivity and switching mechanisms does not result in altering the baseline frequency (second design constraint).

C. Planaria Overall Architecture

Figure 10 illustrates the overall architecture of our proposed accelerator, Planaria. As shown, the original monolithic systolic array has been broken down into 16 omni-directional systolic subarrays, where a group of four subarrays form one Fission Pod that contains a Pod Memory. All these 16 subarrays are connected globally along the accelerator chip via the afore-mentioned bi-directional ring busses for input activations and partial sums data movement. Hence, in one extreme, all these ring busses can be switched on to construct the biggest logical accelerator, running only one DNN on the entire accelerator. Alternatively, in another extreme, all of the ring buses can be switched off to provide 16 standalone logical accelerators, spatially co-locating 16 different DNNs simultaneously for multi-tenant execution. Overall, this architecture supports 65 fission scenarios that can simultaneously co-locate various number of DNNs from 1 to 16. Each of the four Fission Pods is connected to one off-chip memory channel. The bus that brings the data from off-chip memory channel simply goes around the subarrays and can fill their weight buffers. This bus is also connected to the Pod Memory to load/store intermediate activations/output to/from the off-chip memory channel. This bus is pipelined and is no different than the bus that feeds the off-chip data to a conventional systolic array. To avoid clutter, Figure 10 does not illustrate the off-chip memory buses. The Fission Pods are connected to their neighbors through a direct link that can foster data reuse to reduce costly off-chip accesses. If data is present in one of the pods, it can be sent to another at most with two hops.

Planaria can fission up to 16 logical accelerators and therefore, it can simultaneously co-locate 16 different DNNs. However, depending on the combination of the co-located DNNs, 65 total fission scenarios are possible. A logical

accelerator, which represents one of these 65 possibilities, can encompass multiple physical Fission Pods. A logical accelerator can either work as a logical monolithic systolic array or further fission if a DNN layer benefits from coarse-grain parallelism. Planaria's interconnections and bus are designed such that, a logical accelerator can take a portion of a Fission Pod and another logical accelerator takes the rest. In Figure 10, one logical accelerator that accelerates DNN_A can comprise the subarrays in Fission Pod-0 with two subarrays from Fission Pod-3 (Fission Pod-3.SystolicSubarray-0 and Fission Pod-3.SystolicSubarray-1). The remaining two subarrays from Fission Pod-3 can form another logical accelerator to accelerate DNN_B.

Dynamic reconfiguration for fission and multi-tenant execution. Conventional systolic arrays operate in tile granularity. That is, they fetch a tile of weights and activations and produce a tile of intermediate activations or outputs. Planaria does not deviate from this convention. Consider a scenario where three DNNs are simultaneously co-located with some fission scheme on Planaria, and fourth DNN is now dispatched to be accelerated. In this case, Planaria allows the old three co-located DNNs finish computing the tile that they are processing. In the meantime, the scheduler decides the new allocation of the subarrays considering the newly dispatched DNN. At the same time, Planaria loads this new fission configuration as a set of bits that decides the direction of the subarrays and the off/on connectivity state of the buses. Each Planaria subarray requires two 6-bit registers, one retaining the current configuration state and the other pre-holding the next state. Six bits is sufficient for reconfiguration of each subarray and its directions/buses. Two bits determines the direction of input activation and partial sums. Each subarray can potentially connect to four other subarrays, which can be in the neighboring Fission Pods, determined by four bits. The direction of connectivity can be deduced from the direction of the subarray. Another eight bits determine the connectivity of the Pod Memory buffers to the subarrays in the same Fission Pod. Similar to conventional systolic design, each subarray is equipped with an instruction buffer and a Program Counter, indicating the current macro instruction. While the subarray is draining the instructions for the old DNNs, Planaria fetches the next instructions associated with the new configuration. The mechanism is no different than prefetching the instructions for a new tile in conventional systolic arrays. The difference is that, each subarray has a designated PC and a designated 4 KB instruction buffer.

Compilation for Planaria. For INFaaS, since each DNN will serve unbounded set of inference requests, it is intuitive to precompile the DNN and run the precompiled binary again and again. Figure 11(a) illustrates the workflow of Planaria compiler. As the DNN may be allocated different number of subarrays (from 1 to 16) during its execution on Planaria, the compiler generates a total of 16 binaries and 16 configuration tables per DNN to cover all the possibilities. The compiler is aware of all the possible architecture fission configurations at compile time. As such, it iterates over all possible configurations to identify the optimal fission configuration as well as the corresponding
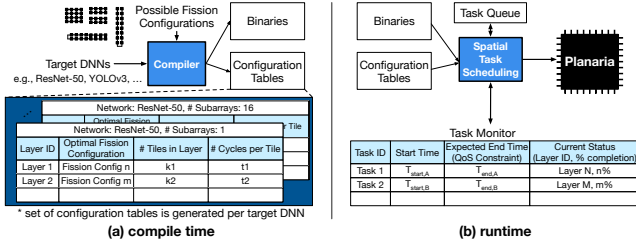
Fig. 11: Overall workflow.

tiling sizes. Importantly, as different layers vary in parallelism and data reuse, so does the optimal fission configuration for the layer. Therefore, for each layer, *in an offline manner*, the configuration table stores the optimal fission configuration, the number of tiles, and the estimated number of cycles per tile. Such estimation is viable because dataflow graphs of DNNs are fixed, there is neither control flow speculation, nor hardware manged cache to cause significant variation in the latency. At runtime, the proposed scheduler which runs on the host CPU uses this software table of estimates to perform QoS-aware scheduling and resource allocation.

V. SPATIAL TASK SCHEDULING

Dynamic architecture fission adds a new dimension in task scheduling, and provides opportunity to break the DNN accelerator into multiple logical accelerators to not only co-locate multiple tasks, but also to provide logical accelerators tailored for the needs of DNN tasks to promote utilization and consequently throughput, SLA satisfaction rate, and fairness. To that end, the scheduler needs to take into account the following requirements:

1) The scheduler ideally needs to be aware of the optimal fission configurations for DNN tasks to leverage dynamic fission and co-location.
2) The scheduler needs to be *QoS-aware* and leverage the available slack time offered by QoS constraint of each task to maximize the co-location and utilization while adhering to the SLA.
3) Task re-allocation requires checkpointing the intermediate results, while making sure that the re-allocation and checkpointing does not overuse on-chip memory or result in significant context switching overheads.

With these requirements, this section delineates the overall flow of our proposed spatial task scheduling (Algorithm 1).

Overall flow. To leverage the dynamic architecture fission, the scheduler is invoked whenever (1) a new inference task is dispatched to the task queue ($\mathcal{Q}$ in Algorithm 1) of the datacenter node or (2) a running inference task finishes. Each scheduling event consists of the following two major stages. Given the DNNs in the queue, the first stage determines the minimum amount of resource (number of subarrays) necessary to meet the QoS requirements for each task. Given that, the second stage determines the allocation of the subarrays based on their availability and priority of the inference requests. This high level flow of the scheduling is shown in function SCHEDULETASKSSPATIALLY.

Estimating minimal resource to meet the QoS requirement. This algorithm exploits the dynamic architecture fission by adaptively assigning resources with regard to the intrinsic slack times provided by each DNN inference task. The algorithm begins by first identifying the minimal resources required to meet the QoS requirement. As illustrated in Figure 11(b), spatial task scheduler utilizes a task monitor to keep track of the running tasks. As shown, the configuration tables generated during compilation is used in conjunction with the current status of the task to predict its remaining time. Thus, the PREDICTTIME function reduces to merely looking up the number of remaining tiles with their cycles and performing simple calculation. Then, the scheduler uses the prediction for each configuration to determine the minimum number of subarrays for each task. The ESTIMATERESOURCES function in Algorithm 1 summarizes this stage.

Allocating resources to improve QoS. After identifying minimal resource for each task, the scheduler determines whether all the tasks in the queue can be co-located simultaneously. Depending on whether or not all the tasks can be spatially co-located on Planaria, this stage invokes two different functions, ALLOCATEFITTASKS and ALLOCATEUNFITTASKS, as shown in line 6–10 in Algorithm 1. First, when all the tasks can be spatially co-located, the function ALLOCATEFITTASKS will first assign the minimum number of subarray required to meet the QoS requirements. Then, if there are remaining resources, the scheduler aims to optimally distribute these spare resources using a score function that balances priority and the remaining time of each task, as shown in line 27 in Algorithm 1. Consequently, this score function not only fosters throughput but also the fairness among the tasks. Finally, the scheduler allocates the spare resources proportional to the score of each task.

On the other hand, when only subset of the tasks fit on Planaria, the scheduler uses the ALLOCATEUNFITTASKS function to resolve the competition among the tasks. Similar to the approach used to assign the spare resources, the function leverages a score that uses priority, slack, and the minimum required resource of each task, as shown in line 40 in Algorithm 1. This scoring mechanism gives advantages to the tasks with higher priority to improve fairness, and to the ones with less slack time or less resource requirement to maximize QoS satisfaction and throughput. Finally, the scheduler allocates the resources to different tasks in the order of their scores until Planaria becomes fully occupied.

Tile-based scheduling to minimize re-allocation overheads. To prevent the running tasks from stalling, (1) the scheduling happens at tile-granularity and (2) the tasks are preempted only when the resource allocation changes. These two strategies in tandem minimizes the potential preemption delays that may reduce throughput. Moreover, the tile-based scheduling minimizes the memory requirements for preemption as only a single tile of intermediate results needs to be stored off-chip. This in turn obviates the need for additional on-chip storage to support preemption.

Algorithm 1 Spatial Scheduling for Planaria

```
 1: function SCHEDULETASKSSPATIALLY(Q):
 2:     estimates ← {}
 3:     for task in Q do
 4:         estimates[task] ← ESTIMATERESOURCES (task)
 5:     end for
 6:     if Planaria.fits(estimates) then
 7:         s ← ALLOCATEFITTASKS (Q, estimates)
 8:     else
 9:         s ← ALLOCATEUNFITTASKS (Q, estimates)
10:     end if
11:     return s
12: end function

13: function ESTIMATERESOURCES(task):
14:     candidates ← [ ]
15:     slack = task.constraint - task.executed_time
16:     for num_subarray in range(Planaria.size) do
17:         if PREDICTTIME(task) ≤ slack then
18:             candidates.append(num_subarray)
19:         end if
20:     end for
21:     return min(candidates)
22: end function

23: function ALLOCATEFITTASKS(Q, estimates):
24:     allocation ← {}, scores ← {}
25:     for task in Q do
26:         allocation[task] ← estimates
27:         scores[task] ← task.priority / task.remaining_time
28:     end for
29:     remaining_array ← Planaria.size - ∑estimates
30:     for task in Q do
31:         fraction ← scores[task] / ∑scores
32:         allocation[task] += fraction × remaining_array
33:     end for
34:     return allocation
35: end function

36: function ALLOCATEUNFITTASKS(Q, estimates):
37:     allocation ← {}, scores ← {}
38:     for task in Q do
39:         slack ← task.constraints - task.executed_time
40:         scores[task] ← task.priority / (slack × estimates[task])
41:     end for
42:     scores.sort(reversed=True)
43:     remaining_array ← Planaria.size
44:     while remaining_array > 0 do
45:         allocation[task] ← estimates[task]
46:         remaining_array -= estimates[task]
47:     end while
48:     return allocation
49: end function
```

Workload	Load Weight	Domain	DNN Model (Release year)
Workload Scenario-A	Heavier	Image Classification	ResNet-50 (2015), GoogLeNet (2014)
		Object Detection	YOLOv3 (2018), SSD-R (2016)
		Machine Translation	GNMT (2016)
Workload Scenario-B	Lighter	Image Classification	EfficientNet-B0 (2019), MobileNet-v1 (2017)
		Object Detection	SSD-M (2017), Tiny YOLO (2017)
Workload Scenario-C	Mixed	Image Classification	ResNet-50, GoogLeNet, EfficientNet-B0, MobileNet-v1
		Object Detection	YOLOv3, SSD-ResNet34, SSD-MobileNet, Tiny YOLO
		Machine Translation	GNMT

TABLE I: Workload scenarios and benchmark DNNs from three domains: image classification [56–59], object detection [60–62], and machine translation [63].

Multi-tenant workloads. Commensurate with MLPerf, as Table I shows, we create three INFaaS workload scenarios made up of inference requests to the benchmark DNNs: (a) Workload-A (from requests to ResNet-50 [56], GoogLeNet [57], YOLOv3 [62], SSD-R [60], and GNMT [63]); (b) Workload-B (from requests to EfficientNet-B0 [58], MobileNet [59], SSD-M [60], and Tiny YOLO [61]; and (c) mixed weight Workload-C (from request to all the nine DNNs). To generate multi-tenant instances from these scenarios, we assign a random arrival time for each request from a Poisson distribution, commensurate with MLPerf and other works [64–66] to mimic task dispatching in datacenters. We assign priority levels within the range of 1 to 11 according to [67] to the dispatched tasks from a uniform distribution. We use Quality of Service (QoS) constraints presented by MLPerf for the server scenarios. To well exercise our proposed system, we use three levels of QoS for each workload scenario, (a) QoS-S as a soft QoS constraint (defined as $1\times$ QoS given in MLPerf), (b) QoS-M as a medium constraint ($\frac{1}{4}\times$ QoS), and (c) QoS-H as a hard constraint ($\frac{1}{16}\times$ QoS) to evaluate sensitivity to QoS latency constraints.

Hardware modeling. We implement the proposed omni-directional systolic subarray and the bussing systems including crossbars for the Fission Pods in Verilog and synthesize them with Synopsys Design Compiler (L-2016.03-SP5) using FreePDK-45nm standard cell library [68] to extract their power/area. We model the on-chip SRAM using CACTI-P [69] that provides energy and area. The on-chip busing system is modeled using McPAT 1.3 [70] and the energy cost estimated to be 0.64 pJ/bit per hop.

Simulation infrastructure for Planaria. We compile each DNN benchmark to Planaria, and develop a cycle-accurate simulator that provides the cycle counts and statistics for energy measurements for each DNN using the modeling described above. We include all the overheads of reconfiguration, fission, instruction fetch, off-chip memory accesses, etc. We verify the cycle counts with our Verilog implementations.

Comparison with PREMA. We compare our proposed Planaria accelerator that supports spatial multi-tenant execution of DNNs to PREMA [4] that offers multi-tenancy via temporal execution. Baseline PREMA utilizes a monolithic TPU-like systolic DNN accelerator as its hardware. For fair comparison, we use the same number of PEs (128×128=16,384), on-chip activation/weight/output buffers (12 MB), frequency (700 MHz),

VI. EVALUATION

A. Methodology

Benchmark DNNs. Following the MLPerf [49] methodology, we choose our representative DNN models from domains of image classification [56–59], object detection [60–62], and machine translation [63]. We use nine diverse DNNs from these domains to construct a set of DNN tasks with various layer dimensions and types of operations including recent and state-of-the-art deep neural models such as EfficientNet and YOLOv3.

and off-chip memory bandwidth as reported in PREMA. The detailed analysis of the synthesis results shows that our design can meet 1GHz frequency and the added omni-directional links or the buses are not on the critical path due to pipelining. However, for fair comparison with PREMA [4], we still use their reported 700 MHz frequency which is based on TPU [1].

PREMA's monolithic systolic array is not explicitly optimized to execute the most recent DNNs that use depth-wise convolutions such as EfficientNet and MobileNet. For this reason, in our evaluation, Workload-A does not include any DNNs with separable depth-wise convolutions. However, it is most reasonable to expect both heavy and lightweight workloads running on the same accelerator, according to industry collaborators. For instance, Google Photos runs image classification (e.g., GoogLeNet), object detection (e.g., MobileNet), and text recognition (e.g., GNMT) all on the same accelerator. One of Planaria's non-tangible benefits is its adaptability to various DNNs that is not available in monolithic designs. Moreover, the "ligh" and "heavy" are merely MLPerf terminologies. Even light benchmarks such as MobileNet-v1 require 1.1 billion operations/4.2 million parameters. We, in good faith, segregate these DNNs to eliminate any bias in our comparisons.

Evaluation Metrics. To evaluate the effectiveness of the proposed solutions, we use the following metrics:

• **Throughput** is defined as the maximum queries-per-second ($\frac{1}{\lambda}$) achieved by the system according to the Poisson distribution (λ) while meeting the SLA for different QoS constraints (QoS-S, QoS-M, and QoS-H). According to MLPerf [49], meeting SLA is defined as executing an image classification or object detection task 99% of the time and a translation task (e.g. GNMT) 97% of time within its QoS latency bound in a multi-tenant workload. This is the main metric for evaluation of server scenarios for inference tasks in MLPerf [49].

• **SLA Satisfaction Rate** is the fraction of multi-tenant workloads that adheres to the SLA described above.

• **Fairness** measures the equal progress of the tasks while considering task priorities. We use the same definition for fairness given in PREMA baseline [4], as: $fairness = min_{i,j} \frac{PP_i}{PP_j}$, while $PP_i = \frac{T_i^{isolated}}{T_i^{multi-tenant}} / \frac{Priority_i}{\sum Priority_k}$.

• **Energy reduction** compares total energy consumption to run multi-tenant workloads on both Planaria and PREMA.

B. Experimental Results

1) Comparison with PREMA

Throughput comparison. Figure 12 compares the throughput of Planaria with PREMA across various workload scenarios and QoS requirements. For Workload-C as the most comprehensive workload scenario that encompasses all the benchmark DNNs, Planaria improves the throughput by $7.4\times$, $7.2\times$, and $12.2\times$, for QoS-S, QoS-M, and QoS-H, respectively. For Workload-B the improvements increase to $13.2\times$ and $43.1\times$ for QoS-S and QoS-M, respectively, while for QoS-H, the baseline PREMA does not meet the 99% QoS constraints. This trend emanates from the fact that the DNNs in Workload-B include separable depth-wise/point-wise convolutions (except for Tiny YOLO).

Since Planaria has fission capability, it can better utilize its resources for depth-wise convolution while a monolithic design in PREMA cannot conform to the requirements of this layer. This is an additional advantage of fission that enables running these recent DNNs more efficiently. With regard to Workload-A, Planaria improves the throughput by $1.1\times$, $1.5\times$, $2.3\times$, for QoS-S, QoS-M, QoS-H, respectively. These DNNs do not include depth-wise convolution, yet our hardware and scheduling yields significant benefits. Across all three workload scenarios, improvements are more significant for the case of hard QoS. Planaria performs better than PREMA in meeting the stricter QoS requirements, as its scheduler is QoS-aware and allocates resources to tasks based on their QoS.

SLA satisfaction rate comparison. Figure 13 illustrates the SLA satisfaction rate of Planaria and PREMA for a the same throughput ($\frac{1}{\lambda}$). As the results show, Planaria improves the SLA satisfaction rate across all the workloads and QoS requirements. The Planaria's *fission-capable* microarchitecture combined with its *QoS-aware* task scheduling algorithm enables significantly larger number of workloads to be executed while adhering to SLA, compared to PREMA. Based on the adopted QoS constraints from [49], Workload-A allows relatively larger slack time compared to other workloads. As such, both Planaria and PREMA performs relatively better in SLA satisfaction for Workload-A. Except for the case of QoS-S, where both Planaria and PREMA satisfy the SLAs 99% of the time, Planaria provides a 14% and 28% increase in SLA satisfaction rate compared to PREMA. For the case of Workload-B that requires tighter QoS as compared to Workload-A, improvements increase to 22%, 31%, and 51%, for QoS-H, QoS-M, and QoS-S, respectively. Finally, the improvements ranges from 16% to 45% for QoS-S to QoS-H, with respect to the mixed Workload-C.

Fairness comparison. Figure 14 shows fairness with Planaria normalized to fairness with PREMA across all the three workload scenarios. Planaria significantly improves fairness for Workload-A by $2.8\times$, $5.1\times$, $2.8\times$ across the three QoS requirements. Overall, Planaria improves fairness significantly with minimum of $1.9\times$ for (Workload-C, QoS-H) and maximum of $9.1\times$ for (Workload-B, QoS-M). That is because spatial co-location in Planaria allows multiple tasks to progress simultaneously, whereas in temporal co-location only one task is privileged to be executed at a time. Besides, spatial co-location takes advantage of existing underutilized resources to improve execution of other tasks. In addition to that, Planaria's task scheduling algorithm (functions ALLOCATEFITTASKS and ALLOCATEUNFITTASKS in Algorithm 1) ensures that each dispatched task receives adequate number of subarrays with respect to its priority and overall execution time.

Energy comparison. Figure 15 compares the total energy consumption for the execution of workloads on Planaria and PREMA systems. For Workload-A, Planaria consumes slightly more energy than PREMA ranging 11% (QoS-M) to 25% (QoS-S). Multi-tenancy leverages the slack in QoS requirements and as such runs the application slightly slower than an isolated mode to improve throughput and fairness. This slower execution
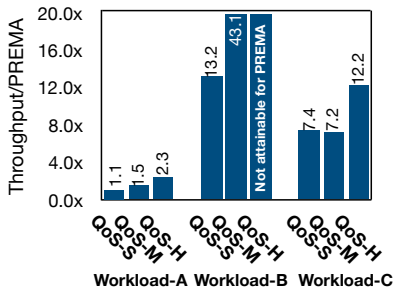
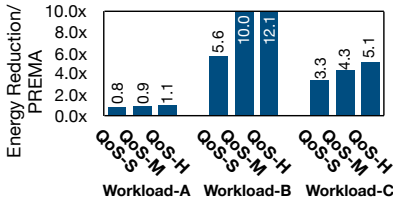Fig. 12: Throughput improvement over PREMA.

Fig. 13: SLA satisfaction rate comparison.

Fig. 14: Fairness improvement over PREMA.

Fig. 15: Planaria energy reduction compared to PREMA.
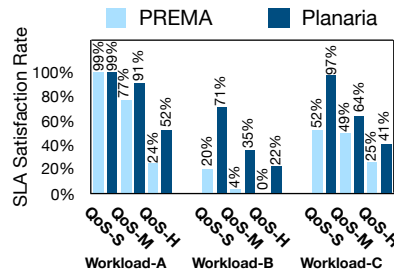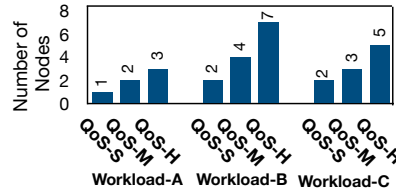

Fig. 16: Required number of nodes to achieve 99% SLA satisfaction. PREMA is not designed for SLA. To avoid unfairness, the results for PREMA is omitted.
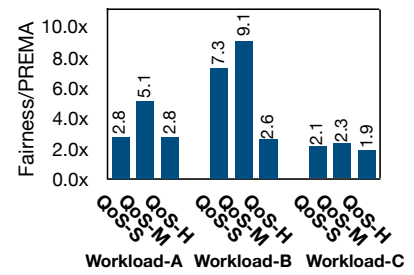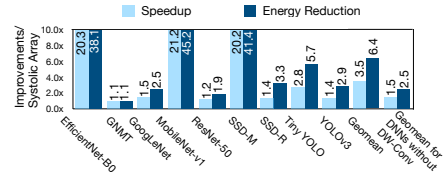

Fig. 17: Planaria improvements for single DNN inference compared to a conventional systolic accelerator with the same on-chip memory and compute resources.

manifests itself as increased total energy compared to running each DNN in isolation with fastest possible speed without considering QoS. As a result, we see a degree of total energy increase for these traditional workloads. In the case of Workload-B and Workload-C, however, when modern DNNs are mixed, the energy benefits from fission outweighs this effect. Workload-B enjoys the maximum energy improvements using Planaria, with minimum of $5.6\times$ and maximum of $12.1\times$ gains over PREMA. A subset of the DNNs in Workload-B require depth-wise convolution layers. Planaria's fission capability significantly reduces the underutilization that monolithic systolic designs suffer due to these layers. Hence, this increase in utilization leads to a higher speedup and lower energy consumption for this workload. More details are presented in Section VI-B2. Overall, with respect to Workload-C which is a mixture of both DNN classes, Planaria reduces the total energy consumption of the workloads by $3.3\times$, $4.3\times$, and $5.1\times$ for QoS-S, QoS-M and QoS-H, respectively.

Scaling out resources. Figure 16 illustrates the minimum number of Planaria nodes necessary to achieve 99% SLA satisfaction, using a constant throughput across all workloads and QoS requirements. In this scaled-out setting, the DNN task traffic is distributed across multiple Planaria-equipped node, where each node has one accelerator. Each DNN task is mapped to a single ship instead of being distributed across multiple nodes. As illustrated in the figure, the number of nodes necessary to achieve SLA satisfaction increases as we go from soft (QoS-S) to hard constraints (QoS-H) on QoS. Among the workload scenarios, Workload-B, which has stricter QoS constraints, requires larger number of nodes compared to other workloads, with minimum of 2 nodes for QoS-S and maximum of 7 nodes for QoS-H. Also, one Planaria accelerator is sufficient for QoS-S of Workload-A which already satisfies the SLAs 99% of the time (Figure 13) and thus obviates the need for an increased number of nodes, whereas QoS-H requires three nodes. Finally, with regard to Workload-C, 2, 3, and 5

nodes are required for QoS-S, QoS-M, andQoS-H, respectively.

2) Sensitivity Studies

Planaria performance/energy on a single DNN inference. Figure 17 shows the speedup and energy reduction of Planaria as compared to a conventional systolic-based accelerator (similar to PREMA's) with the same amount of compute and memory resources, while each DNN inference is executed in isolation. Across the nine DNN benchmarks, Planaria offers $3.5\times$and $6.3\times$ speedup and energy reduction, respectively. The fission-capable design of Planaria enables it to adapt to the various computational characteristics that exist in DNN layers and exploits the opportunities for parallelism and data reuse to improve the performance and energy consumption.

Among them, EfficientNet-B0, MobileNet-v1, and SSD-M which exploit depth-wise convolutions enjoy the maximum benefits. To run depth-wise layers on monolithic systolic arrays, a depth-wise 2-D filter is vectorized and mapped to one column of the array. Lack of input reuse in depth-wise convolution leads to utilizing only one column of the array. This column then accumulates the results of the multiplication of depth-wise filter and its corresponding inputs while the filter weight remains stationary for all the inputs of the pertinent channel. Architecture fission capability and dynamic reconfigurability of Planaria enables it to fission into 16 *independent* smaller subarrays for executing the depth-wise layers. As such, 16 systolic columns, each of which from different subarrays, are utilized to process 16 channels in *parallel* for depth-wise convolution, yielding up to $16\times$ higher utilization. Therefore, the proposed architecture fission yields significantly higher performance for depth-wise convolution.

With regard to DNNs without depth-wise convolution, Tiny YOLO achieves the maximum benefits, $2.8\times$ speedup and $5.5\times$ energy reduction. GNMT attains the least improvements, since it mostly requires matrix-multiplication operations, which is also suitable for a monolithic design. Unlike the multi-tenant case for Workload-A, there is no increase in energy for its isolated
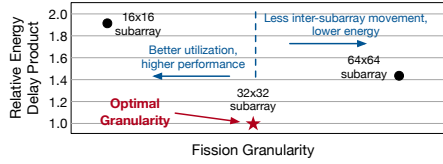
Fig. 18: Design space exploration for fission granularity.

DNNs. As discussed, multi-tenancy trades off individual energy and speed for higher throughput. In the isolated case, that trade-off is not employed and all the resources are allocated to one DNN maximizing its efficiency and speed through fission.

Design space exploration for fission granularity. To find the optimal fission granularity, we perform a design space exploration that yields the most efficient granularity, as shown in Figure 18. We consider 128×128 total number of PEs (as was in PREMA[4] and TPU [1]) and sweep the size of subarrays for 16×16, 32×32, and 64×64. To find the optimal size, we consider Energy-Delay-Product (EDP) and measure its average value across the benchmark DNNs, while they run in isolation. Figure 18 illustrates the relative EDP values for the three design points. Blue arrows in Figure 18 show the tradeoff between energy and performance for the design space exploration with respect to the fission granularity. As Figure 18 shows, 32×32 PEs per subarray offers least EDP, that considers both energy and performance. This is the size that Planaria has adopted for its fission granularity.

Sensitivity analysis for fission possibilities. Table II illustrates the DNN layers sensitivity to various fission possibilities, where the whole accelerator is dedicated to one DNN inference. The dark blue cells of the table show the 15 most fitting fission possibilities for the benchmarks when run in isolation. The table also reports their architectural characteristics (parallelism (P), input activation reuse (IAR), partial sum reuse (PSR), and usage of omni-directional systolic movement (OD-SA)) with respect to the 32×32 fission granularity. A cell also lists the DNNs with the percentage of their layers that have utilized the pertinent fission configuration. Omni-directional systolic design enables six of these configurations. The black cell in Table II captures the most prevalent and fruitful fission configuration that, in fact, exploits the omni-directional feature. All nine DNNs utilize this configuration in their execution, where GNMT, YOLOv3, and MobileNet-v1 are the three DNNs that utilize this configuration more than others. Another important configuration is where fission takes place at the finest granularity and 16 of 32×32 subarrays work independently in parallel. This configuration is specifically important and useful for DNNs with depth-wise convolution, e.g. EfficientNet-B0, MobielNet-v1.

Area and power overheads for fission. Figure 19 illustrates the breakdown of area and power with respect to different hardware components in Planaria when synthesized at 45 nm, without considering on-chip buffers that are the same as one used in PREMA. The breakdown includes the components added to support dynamic fission, which includes the logic for Omni-directional flow of data, Fission Pod crossbar,
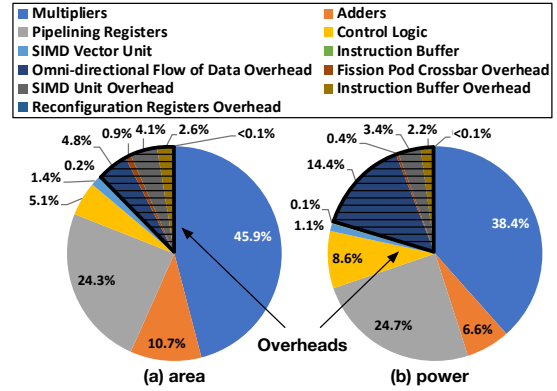

Fig. 19: Planaria power/area breakdown and its overheads.

SIMD vector unit additions, instruction buffer additions, and re-configurations registers. Other components are multipliers, adders and accumulators, pipelining registers for intra-systolic array/subarray data movement, a SIMD unit, control logic, and an instruction buffer. Note that these components are the same for both regular systolic array and Planaria, and consequently these are not considered overheads. Overall, dynamic fission adds 12.6%, 20.6% extra area and power, respectively.

VII. RELATED WORK

The need for higher speed and efficiency in DNN execution has led to an explosion of DNN accelerators [1, 3, 71–93] that has even made their way to operational datacenters (Google's TPU [1], NVIDIA T4 [2], Microsoft Brainwave [3], etc.). However, multi-tenancy has been largely omitted in the proposed or deployed designs due to the arms race in the market for higher speed and efficiency. This paper offers spatial multi-tenant acceleration through architecture fission that is propelled by unique microarchitectural mechanisms and organizations that enables flexible task scheduling. As such, this paper lies at the intersection of DNN acceleration and multi-tenant execution. We discuss relevant related work categories below.

Multi-tenancy for DNN accelerators. PREMA [4] develops a scheduling algorithm for preemptive execution of DNNs on a monolithic accelerator and uses time-sharing for multi-tenancy. On the other hand, AI-MT [94] develops an architecture that supports multi-tenancy by first tiling the layers at compile time, then exploiting hardware-based scheduling to maximize resource utilization. In contrast to these temporal multi-tenancy supports, this paper explores architecture design for spatial co-location of DNNs for multi-tenant acceleration and its unique scheduling challenges.

Flexibility in DNN accelerators. Flexibility in DNN acceleration has recently gained attention [52–54, 88, 95, 96]. However, these inspiring works do not explore simultaneous spatial co-location of multiple DNNs on the same chip. Eyeriss v2 [54] proposes a hierarchical architecture equipped with a flexible mesh-based NoC that provides flexibility to adapt to various level of data reuse. MAERI [52] and SIGMA [95] propose a reconfigurable interconnect among the PEs to deal with sparsity in neural networks [52] and matrix multiplications [95] and to

(128x128)-1 Cluster — P: 1x, IAR: 4x, PSR: 4x, OD-SA: Unused

DNN	% of Layers
EfficientNet-B0	7.3 %
GoogLeNet	15.5 %
MobileNet-v1	7.1 %
ResNet-50	26.5 %
SSD-M	26.1 %
SSD-R	62.5 %
Tiny YOLO	11.1 %
YOLOv3	21.2 %

(32x512)-1 Cluster — P: 1x, IAR: 16x, PSR: 1x, OD-SA: Used

DNN	% of Layers
EfficientNet-B0	15.9 %

(512x32)-1 Cluster — P: 1x, IAR: 1x, PSR: 16x, OD-SA: Used

DNN	% of Layers
EfficientNet-B0	9.8 %
GoogLeNet	15.5 %
ResNet-50	6.1 %
Tiny YOLO	22.2 %
YOLOv3	7.7 %

(64x256)-1 Cluster — P: 1x, IAR: 8x, PSR: 2x, OD-SA: Used

DNN	% of Layers
EfficientNet-B0	11.0 %
GoogLeNet	1.7 %
ResNet-50	8.2 %

(256x64)-1 Cluster — P: 1x, IAR: 8x, PSR: 8x, OD-SA: Used

DNN	% of Layers
EfficientNet-B0	7.3 %
GNMT	100 %
GoogLeNet	29.3 %
MobileNet-v1	35.7 %
ResNet-50	32.6 %
SSD-M	26.1 %
SSD-R	16.7 %
Tiny YOLO	22.2 %
YOLOv3	53.8 %

(32x256)-2 Clusters — P: 2x, IAR: 2x, PSR: 1x, OD-SA: Used

DNN	% of Layers
EfficientNet-B0	2.4 %
YOLOv3	1.9 %

(256x32)-2 Clusters — P: 2x, IAR: 8x, PSR: 8x, OD-SA: Used

DNN	% of Layers
EfficientNet-B0	1.2 %
GoogLeNet	12.1 %
Tiny YOLO	11.1 %

(64x128)-2 Clusters — P: 2x, IAR: 4x, PSR: 2x, OD-SA: Unused

DNN	% of Layers
GoogLeNet	6.9 %
MobileNet-v1	3.6 %
SSD-M	4.3 %
SSD-R	4.2 %
YOLOv3	5.8 %

(128x64)-2 Clusters — P: 2x, IAR: 2x, PSR: 4x, OD-SA: Unused

DNN	% of Layers
GoogLeNet	3.4 %
ResNet-50	12.2 %
YOLOv3	5.8 %

(32x128)-4 Clusters — P: 4x, IAR: 4x, PSR: 1x, OD-SA: Unused

DNN	% of Layers
EfficientNet-B0	1.2 %
SSD-M	8.7 %

(128x32)-4 Clusters — P: 4x, IAR: 1x, PSR: 4x, OD-SA: Unused

DNN	% of Layers
GoogLetNet	8.6 %
SSD-M	4.3 %

(64x64)-4 Clusters — P: 4x, IAR: 2x, PSR: 2x, OD-SA: Unused

DNN	% of Layers
EfficientNet-B0	1.2 %
ResNet-50	8.2 %
SSD-R	16.7 %
Tiny YOLO	11.1 %
YOLOv3	1.9 %

(64x32)-8 Clusters — P: 8x, IAR: 1x, PSR: 2x, OD-SA: Unused

DNN	% of Layers
GoogLeNet	1.7 %
Tiny YOLO	11.1 %
YOLOv3	1.9 %

(32x64)-8 Clusters — P: 8x, IAR: 2x, PSR: 1x, OD-SA: Unused

DNN	% of Layers
EfficientNet-B0	2.4 %
MobileNet-v1	7.1 %
SSD-M	4.3 %

(32x32)-16 Clusters — P: 16x, IAR: 1x, PSR: 1x, OD-SA: Unused

DNN	% of Layers
EfficientNet-B0	23.1 %
GoogLeNet	1.7 %
MobileNet-v1	46.4 %
ResNet-50	6.1 %
SSD-M	26.1 %
Tiny YOLO	11.1 %

TABLE II: Layer sensitivity to various fission configurations. Each cell shows a configuration with its architectural attributes (parallelism, input activation reuse, partial sum reuse, and usage of omni-directional data movement) and the percentage of the layers that uses the configuration.

increase resource utilization. Simba [96] proposes a scalable multi-chip module-based accelerator to reduce fabrication cost and provide scalability with respect to inter-chip and intra-chip communication. BitFusion [88] explores bit-level dynamic composability in its multipliers to support heterogeneity in deeply quantized neural networks. Tangram [53] explores dataflow optimizations by buffer-sharing dataflow and inter-layer pipelining on a hierarchical design to reduce energy. **Multi-tenancy for CPUs and GPUs.** There is a large swath of related work on multi-tenancy for CPUs [36, 66, 97–103] and GPUs [29, 35, 37–42, 66, 104–110] due to its vitality for cloud-scale computing. NVIDIA Triton Inference Server [111] (formerly TensorRT Inference Server) provides a cloud software inference solution optimized for GPUs and offers benefits by supporting multi-tenant execution of DNNs on them [112]. GrandSLAm [66] proposes scheduling policies to minimize SLA violation rates for microservices in the cloud for CPUs and GPUs, and the studied workloads include DNNs. In contrast, this paper uniquely enables spatial multi-tenancy on DNN accelerators, by leveraging a dynamic fission in the architecture and leveraging that through the scheduler. Kubernetes [48] and Mesos [113] are cloud-scale resource management framework, but have not explored spatial multi-tenancy in DNN accelerators due to its unavailability. Our scheduling algorithm is complementary to their operation. **DNN acceleration.** There is a large body of work [1, 3, 52–55, 71–82, 84–88, 91–93, 95, 96, 114–124] for isolated acceleration of DNNs that, although inspired our work, are not focused on multi-tenancy but rather offer various innovations to improve the speed and efficiency of DNN execution.

VIII. Conclusion

As INFerence-as-a-Service is growing in demand, it is timely to explore multi-tenancy for DNN accelerators. This paper explored this topic through a novel approach of dynamic architecture fission, and provided a concrete architecture, Planaria, and its respective scheduling algorithm. Evaluation with a diverse set of DNN benchmarks and workload scenarios shows significant gains in throughput, SLA satisfaction, and fairness.

IX. Acknowledgement

References

[1] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *ISCA*, 2017.

[2] Nvidia T4: Tensor core GPU for AI inference. https://www.nvidia.com/en-us/data-center/tesla-t4/, .

[3] Jeremy Fowers, Kalin Ovtcharov, Michael Papamichael, Todd Massengill, Ming Liu, Daniel Lo, Shlomi Alkalay, Michael Haselman, Logan Adams, Mahdi Ghandi, et al. A configurable cloud-scale dnn processor for real-time ai. In *ISCA*, 2018.

[4] Yujeong Choi and Minsoo Rhu. Prema: A predictive multi-task scheduling algorithm for preemptible neural processing units. *HPCA*, 2020.

[5] R. H. Dennard, F. H. Gaensslen, V. L. Rideout, E. Bassous, and A. R. LeBlanc. Design of ion-implanted MOSFET's with very small physical dimensions. *JSSC*, 1974.

[6] N. Hardavellas, M. Ferdman, B. Falsafi, and A. Ailamaki. Toward dark silicon in servers. *IEEE Micro*, 2011.

[7] Ganesh Venkatesh, Jack Sampson, Nathan Goulding, Saturnino Garcia, Vladyslav Bryksin, Jose Lugo-Martinez, Steven Swanson, and Michael Bedford Taylor. Conservation cores: Reducing the energy of mature computations. In *ASPLOS*, 2010.

[8] Hadi Esmaeilzadeh, Emily Blem, Renee St. Amant, Karthikeyan Sankaralingam, and Doug Burger. Dark silicon and the end of multicore scaling. In *ISCA*, 2011.

[9] John L Hennessy and David A Patterson. A new golden age for computer architecture. *CACM and Turing Lecture*, 2019.

[10] Rehan Hameed, Wajahat Qadeer, Megan Wachs, Omid Azizi, Alex Solomatnikov, Benjamin C Lee, Stephen Richardson, Christos Kozyrakis, and Mark Horowitz. Understanding sources of inefficiency in general-purpose chips. In *ISCA*, 2010.

[11] Edge TPU. https://cloud.google.com/edge-tpu/.

[12] Nvidia Jetson: The AI platform for autonomous machines. https://developer.nvidia.com/embedded/develop/hardware, .

[13] Apple a11-bionic. https://en.wikipedia.org/wiki/Apple_A11, .

[14] Francisco Romero, Qian Li, Neeraja J Yadwadkar, and Christos Kozyrakis. INFaaS: Managed & model-less inference serving. *arXiv*, 2019.

[15] Google assistant. https://assistant.google.com, .

[16] Apple siri. https://www.apple.com/siri/, .

[17] Amazon alexa. https://developer.amazon.com/en-US/alexa/, .

[18] Google cloud customers. https://cloud.google.com/customers/, .

[19] Amazon case studies. https://aws.amazon.com/solutions/case-studies/, .

[20] Amazon sagemaker customers. https://aws.amazon.com/sagemaker/customers/, .

[21] Google cloud. https://cloud.google.com/products/ai/, .

[22] Amazon elastic inference. https://aws.amazon.com/machine-learning/elastic-inference/.

[23] Amazon sagemaker. https://aws.amazon.com/sagemaker/, .

[24] Azure machine learning. https://azure.microsoft.com/en-us/services/machine-learning/.

[25] Udit Gupta, Samuel Hsia, Vikram Saraph, Xiaodong Wang, Brandon Reagen, Gu-Yeon Wei, Hsien-Hsin S Lee, David Brooks, and Carole-Jean Wu. Deeprecsys: A system for optimizing end-to-end at-scale neural recommendation inference. *ISCA*, 2020.

[26] Yuhao Zhu, Matthew Halpern, and Vijay Janapa Reddi. Event-based scheduling for energy-efficient qos (eqos) in mobile web applications. In *HPCA*, 2015.

[27] Yuhao Zhu, Daniel Richins, Matthew Halpern, and Vijay Janapa Reddi. Microarchitectural implications of event-driven server-side web applications. In *MICRO*, 2015.

[28] Bo Wu, Guoyang Chen, Dong Li, Xipeng Shen, and Jeffrey Vetter. Enabling and exploiting flexible task assignment on gpu through sm-centric program transformations. In *ICS*, 2015.

[29] Guoyang Chen and Xipeng Shen. Free launch: optimizing gpu dynamic kernel launches through thread reuse. In *MICRO*, 2015.

[30] Bo Wu, Xu Liu, Xiaobo Zhou, and Changjun Jiang. Flep: Enabling flexible and efficient preemption on gpus. In *ASPLOS*, 2017.

[31] Harshad Kasture and Daniel Sanchez. Ubik: Efficient Cache Sharing with Strict QoS for Latency-Critical Workloads. In *ASPLOS*, 2014.

[32] Christina Delimitrou, Daniel Sanchez, and Christos Kozyrakis. Tarcil: Reconciling Scheduling Speed and Quality in Large Shared Clusters. In *SoCC*, 2015.

[33] Yuzhao Wang, Lele Li, You Wu, Junqing Yu, Zhibin Yu, and Xuehai Qian. Tpshare: a time-space sharing scheduling abstraction for shared cloud via vertical labels. In *ISCA*, 2019.

[34] Abdulaziz Tabbakh, Murali Annavaram, and Xuehai Qian. Power efficient sharing-aware gpu data management. In *IPDPS*, 2017.

[35] Nandita Vijaykumar, Kevin Hsieh, Gennady Pekhimenko, Samira Khan, Ashish Shrestha, Saugata Ghose, Adwait Jog, Phillip B Gibbons, and Onur Mutlu. Zorua: A holistic approach to resource virtualization in gpus. In *MICRO*, 2016.

[36] Sudipto Das, Vivek R. Narasayya, Feng Li, and Manoj Syamala. Cpu sharing techniques for performance isolation in multi-tenant relational database-as-a-service. *Proc. VLDB Endow.*, 2013.

[37] Quan Chen, Hailong Yang, Jason Mars, and Lingjia Tang. Baymax: Qos awareness and increased utilization for non-preemptive accelerators in warehouse scale computers. *ASPLOS*, 2016.

[38] Quan Chen, Hailong Yang, Minyi Guo, Ram Srivatsa Kannan, Jason Mars, and Lingjia Tang. Prophet: Precise qos prediction on non-preemptive accelerators to improve utilization in warehouse-scale computers. *ASPLOS*, 2017.

[39] Ivan Tanasic, Isaac Gelado, Javier Cabezas, Alex Ramirez, Nacho Navarro, and Mateo Valero. Enabling preemptive multiprogramming on gpus. In *ISCA*, 2014.

[40] Jason Jong Kyu Park, Yongjun Park, and Scott Mahlke. Chimera: Collaborative preemption for multitasking on a shared gpu. *ASPLOS*, 2015.

[41] Jason Jong Kyu Park, Yongjun Park, and Scott Mahlke. Dynamic resource management for efficient utilization of multitasking gpus. *ASPLOS*, 2017.

[42] Dipanjan Sengupta, Anshuman Goswami, Karsten Schwan, and Krishna Pallavi. Scheduling multi-tenant cloud workloads on accelerator-based systems. In *SC*, 2014.

[43] David Wentzlaff, Charles Gruenwald III, Nathan Beckmann, Kevin Modzelewski, Adam Belay, Lamia

Youseff, Jason Miller, and Anant Agarwal. An operating system for multicore and clouds: Mechanisms and implementation. In *SoCC*, 2010.

[44] Prashanth Thinakaran, Jashwant Raj, Bikash Sharma, Mahmut T Kandemir, and Chita R Das. The curious case of container orchestration and scheduling in gpu-based datacenters. In *SoCC*, 2018.

[45] Adwait Jog, Onur Kayiran, Tuba Kesten, Ashutosh Pattnaik, Evgeny Bolotin, Niladrish Chatterjee, Stephen W Keckler, Mahmut T Kandemir, and Chita R Das. Anatomy of gpu memory system for multi-application execution. In *MEMSYS*, 2015.

[46] Prashanth Thinakaran, Jashwant Raj Gunasekaran, Bikash Sharma, Mahmut Taylan Kandemir, and Chita R Das. Kube-knots: Resource harvesting through dynamic container orchestration in gpu-based datacenters. In *CLUSTER*, 2019.

[47] Natalie Enright Jerger, Dana Vantrease, and Mikko Lipasti. An evaluation of server consolidation workloads for multi-core designs. In *IISWC*, 2007.

[48] Kubernetes. https://kubernetes.io.

[49] Vijay Janapa Reddi, Christine Cheng, David Kanter, Peter Mattson, Guenther Schmuelling, Carole-Jean Wu, Brian Anderson, Maximilien Breughe, Mark Charlebois, William Chou, et al. Mlperf inference benchmark. *arxiv*, 2019.

[50] Peter Mattson, Christine Cheng, Cody Coleman, Greg Diamos, Paulius Micikevicius, David Patterson, Hanlin Tang, Gu-Yeon Wei, Peter Bailis, Victor Bittorf, et al. Mlperf training benchmark. *arxiv*, 2019.

[51] Zero-shot translation with google's multilingual neural machine translation system. https://ai.googleblog.com/2016/11/zero-shot-translation-with-googles.html.

[52] Hyoukjun Kwon, Ananda Samajdar, and Tushar Krishna. Maeri: Enabling flexible dataflow mapping over dnn accelerators via reconfigurable interconnects. *ASPLOS*, 2018.

[53] Mingyu Gao, Xuan Yang, Jing Pu, Mark Horowitz, and Christos Kozyrakis. Tangram: Optimized coarse-grained dataflow for scalable nn accelerators. In *ASPLOS*, 2019.

[54] Yu-Hsin Chen, Tien-Ju Yang, Joel Emer, and Vivienne Sze. Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices. *JETCAS*, 2019.

[55] Hyoukjun Kwon, Prasanth Chatarasi, Michael Pellauer, Angshuman Parashar, Vivek Sarkar, and Tushar Krishna. Understanding reuse, performance, and hardware cost of dnn dataflow: A data-centric approach. In *MICRO*, 2019.

[56] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[57] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *CVPR*, 2015.

[58] Mingxing Tan and Quoc V Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *ICML*,

2019.

[59] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv*, 2017.

[60] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *ECCV*, 2016.

[61] Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. In *CVPR*, 2017.

[62] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv*, 2018.

[63] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv*, 2016.

[64] Balajee Vamanan, Hamza Bin Sohail, Jahangir Hasan, and TN Vijaykumar. Timetrader: Exploiting latency tail to save datacenter energy for online search. In *MICRO*, 2015.

[65] Lalith Suresh, Peter Bodik, Ishai Menache, Marco Canini, and Florin Ciucu. Distributed resource management across process boundaries. In *SoCC*, 2017.

[66] Ram Srivatsa Kannan, Lavanya Subramanian, Ashwin Raju, Jeongseob Ahn, Jason Mars, and Lingjia Tang. Grandslam: Guaranteeing slas for jobs in microservices execution frameworks. In *EuroSys*, 2019.

[67] Pascale Minet, Eric Renault, Ines Khoufi, and Selma Boumerdassi. Analyzing traces from a google data center. In *IWCMC*, 2018.

[68] FreePDK45. https://www.eda.ncsu.edu/wiki/FreePDK45.

[69] S. Li, K. Chen, J. H. Ahn, J. B. Brockman, and N. P. Jouppi. CACTI-P: Architecture-level Modeling for SRAM-based Structures with Advanced Leakage Reduction Techniques. In *ICCAD*, 2011.

[70] Sheng Li, Jung Ho Ahn, R.D. Strong, J.B. Brockman, D.M. Tullsen, and N.P. Jouppi. McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *MICRO*, 2009.

[71] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, Ling Li, Tianshi Chen, Zhiwei Xu, Ninghui Sun, et al. Dadiannao: A machine-learning supercomputer. In *MICRO*, 2014.

[72] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. In *ISCA*, 2016.

[73] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. Eie: efficient inference engine on compressed deep neural network. In *ISCA*, 2016.

[74] Jorge Albericio, Patrick Judd, Tayler Hetherington, Tor Aamodt, Natalie Enright Jerger, and Andreas Moshovos.

Cnvlutin: ineffectual-neuron-free deep neural network computing. In *ISCA*, 2016.

[75] Shaoli Liu, Zidong Du, Jinhua Tao, Dong Han, Tao Luo, Yuan Xie, Yunji Chen, and Tianshi Chen. Cambricon: An instruction set architecture for neural networks. In *ISCA*, 2016.

[76] Shijin Zhang, Zidong Du, Lei Zhang, Huiying Lan, Shaoli Liu, Ling Li, Qi Guo, Tianshi Chen, and Yunji Chen. Cambricon-x: An accelerator for sparse neural networks. In *MICRO*, 2016.

[77] Patrick Judd, Jorge Albericio, Tayler Hetherington, Tor M Aamodt, and Andreas Moshovos. Stripes: Bit-serial deep neural network computing. In *MICRO*, 2016.

[78] Ali Shafiee, Anirban Nag, Naveen Muralimanohar, Rajeev Balasubramonian, John Paul Strachan, Miao Hu, R Stanley Williams, and Vivek Srikumar. Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars. In *ISCA*, 2016.

[79] Ping Chi, Shuangchen Li, Cong Xu, Tao Zhang, Jishen Zhao, Yongpan Liu, Yu Wang, and Yuan Xie. Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory. In *ISCA*, 2016.

[80] Duckhwan Kim, Jaeha Kung, Sek Chai, Sudhakar Yalamanchili, and Saibal Mukhopadhyay. Neurocube: A programmable digital neuromorphic architecture with high-density 3d memory. In *ISCA*, 2016.

[81] Brandon Reagen, Paul Whatmough, Robert Adolf, Saketh Rama, Hyunkwang Lee, Sae Kyu Lee, José Miguel Hernández-Lobato, Gu-Yeon Wei, and David Brooks. Minerva: Enabling low-power, highly-accurate deep neural network accelerators. In *ISCA*, 2016.

[82] Mingyu Gao, Jing Pu, Xuan Yang, Mark Horowitz, and Christos Kozyrakis. Tetris: Scalable and efficient neural network acceleration with 3d memory. In *ASPLOS*, 2017.

[83] Renzo Andri, Lukas Cavigelli, Davide Rossi, and Luca Benini. Yodann: An ultra-low power convolutional neural network accelerator based on binary weights. *arXiv*, 2016.

[84] Hardik Sharma, Jongse Park, Divya Mahajan, Emmanuel Amaro, Joon Kim, Chenkai Shao, Asit Misra, and Hadi Esmaeilzadeh. From high-level deep neural models to fpgas. In *MICRO*, 2016.

[85] Angshuman Parashar, Minsoo Rhu, Anurag Mukkara, Antonio Puglielli, Rangharajan Venkatesan, Brucek Khailany, Joel Emer, Stephen W Keckler, and William J Dally. SCNN: An Accelerator for Compressed-sparse Convolutional Neural Networks. In *ISCA*, 2017.

[86] Linghao Song, Xuehai Qian, Hai Li, and Yiran Chen. Pipelayer: A pipelined reram-based accelerator for deep learning. In *HPCA*, 2017.

[87] Jorge Albericio, Alberto Delmás, Patrick Judd, Sayeh Sharify, Gerard O'Leary, Roman Genov, and Andreas Moshovos. Bit-pragmatic deep neural network computing. In *MICRO*, 2017.

[88] Hardik Sharma, Jongse Park, Naveen Suda, Liangzhen Lai, Benson Chau, Vikas Chandra, and Hadi Esmaeilzadeh. Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural networks. *ISCA*, 2018.

[89] Vahide Aklaghi, Amir Yazdanbakhsh, Kambiz Samadi, Hadi Esmaeilzadeh, and Rajesh K. Gupta. Snapea: Predictive early activation for reducing computation in deep convolutional neural networks. In *ISCA*, 2018.

[90] Kartik Hegde, Jiyong Yu, Rohit Agrawal, Mengjia Yan, Michael Pellauer, and Christopher W Fletcher. Ucnn: Exploiting computational reuse in deep neural networks via weight repetition. *arXiv*, 2018.

[91] Jinmook Lee, Changhyeon Kim, Sanghoon Kang, Dongjoo Shin, Sangyeob Kim, and Hoi-Jun Yoo. Unpu: A 50.6 tops/w unified deep neural network accelerator with 1b-to-16b fully-variable weight bit-precision. In *ISSCC*, 2018.

[92] Alberto Delmas Lascorz, Patrick Judd, Dylan Malone Stuart, Zissis Poulos, Mostafa Mahmoud, Sayeh Sharify, Milos Nikolic, Kevin Siu, and Andreas Moshovos. Bit-tactical: A software/hardware approach to exploiting value and bit sparsity in neural networks. In *ASPLOS*, 2019.

[93] Sayeh Sharify, Alberto Delmas Lascorz, Mostafa Mahmoud, Milos Nikolic, Kevin Siu, Dylan Malone Stuart, Zissis Poulos, and Andreas Moshovos. Laconic deep learning inference acceleration. In *ISCA*, 2019.

[94] Eunjin Baek, Dongup Kwon, and Jangwoo Kim. A multi-neural network acceleration architecture. *ISCA*, 2020.

[95] Eric Qin, Ananda Samajdar, Hyoukjun Kwon, Vineet Nadella, Sudarshan Srinivasan, Dipankar Das, Bharat Kaul, and Tushar Krishna. Sigma: A sparse and irregular gemm accelerator with flexible interconnects for dnn training. *HPCA*, 2020.

[96] Yakun Sophia Shao, Jason Clemons, Rangharajan Venkatesan, Brian Zimmer, Matthew Fojtik, Nan Jiang, Ben Keller, Alicia Klinefelter, Nathaniel Pinckney, Priyanka Raina, et al. Simba: Scaling deep-learning inference with multi-chip-module-based architecture. In *MICRO*, 2019.

[97] James E. Smith and Andrew R. Pleszkun. Implementing precise interrupts in pipelined processors. *TC*, 1988.

[98] Lingjia Tang, Jason Mars, and Mary Lou Soffa. Compiling for niceness: Mitigating contention for qos in warehouse scale computers. In *CGO*, 2012.

[99] Jason Mars and Lingjia Tang. Whare-map: heterogeneity in" homogeneous" warehouse-scale computers. In *ISCA*, 2013.

[100] Jason Mars, Lingjia Tang, Robert Hundt, Kevin Skadron, and Mary Lou Soffa. Bubble-up: Increasing utilization in modern warehouse scale computers via sensible co-locations. In *MICRO*, 2011.

[101] Wei Wang, Tanima Dey, Jason Mars, Lingjia Tang, Jack W Davidson, and Mary Lou Soffa. Performance analysis of thread mappings with a holistic view of the

hardware resources. In *ISPASS*, 2012.

[102] Christina Delimitrou and Christos Kozyrakis. Quasar: resource-efficient and qos-aware cluster management. In *ASPLOS*, 2014.

[103] Yunqi Zhang, Michael A Laurenzano, Jason Mars, and Lingjia Tang. Smite: Precise qos prediction on real-system smt processors to improve utilization in warehouse scale computers. In *MICRO*, 2014.

[104] Haishan Zhu and Mattan Erez. Dirigent: Enforcing qos for latency-critical tasks on shared multicore systems. In *ASPLOS*, 2016.

[105] Z. Wang, J. Yang, R. Melhem, B. Childers, Y. Zhang, and M. Guo. Simultaneous multikernel gpu: Multi-tasking throughput processors via fine-grained sharing. In *HPCA*, 2016.

[106] Zhou Fang, Tong Yu, Ole J Mengshoel, and Rajesh K Gupta. Qos-aware scheduling of heterogeneous servers for inference in deep neural networks. In *CIKM*, 2017.

[107] Y. Ukidave, X. Li, and D. Kaeli. Mystic: Predictive scheduling for gpu based cloud servers using machine learning. In *IPDPS*, 2016.

[108] Z. Lin, L. Nyland, and H. Zhou. Enabling efficient preemption for simt architectures with lightweight context switching. In *SC*, 2016.

[109] Vignesh T. Ravi, Michela Becchi, Gagan Agrawal, and Srimat Chakradhar. Supporting gpu sharing in cloud environments with a transparent runtime consolidation framework. In *HPDC*, 2011.

[110] Michela Becchi, Kittisak Sajjapongse, Ian Graves, Adam Procter, Vignesh Ravi, and Srimat Chakradhar. A virtual memory based runtime to support multi-tenancy in clusters with gpus. In *HPDC*, 2012.

[111] Nvidia triton inference server. https://github.com/NVIDIA/triton-inference-server/.

[112] Tripti Singhal. Maximizing gpu utilization for datacenter inference with nvidia tensorrt inference server. 2019.

[113] Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D Joseph, Randy Katz, Scott Shenker, and Ion Stoica. Mesos: a platform for fine-grained resource sharing in the data center. In *NSDI*, 2011.

[114] Sayeh Sharify, Alberto Delmas Lascorz, Kevin Siu, Patrick Judd, and Andreas Moshovos. Loom: Exploiting weight and activation precisions to accelerate convolutional neural networks. In *DAC*, 2018.

[115] Soroush Ghodrati, Hardik Sharma, Sean Kinzer, Amir Yazdanbakhsh, Jongse Park, Nam Sung Kim, Doug Burger, and Hadi Esmaeilzadeh. Mixed-signal charge-domain acceleration of deep neural networks through interleaved bit-partitioned arithmetic. In *PACT*, 2020.

[116] Soroush Ghodrati, Hardik Sharma, Cliff Young, Nam Sung Kim, and Hadi Esmaeilzadeh. Bit-parallel vector composability for neural acceleration. In *DAC*, 2020.

[117] Byung Hoon Ahn, Prannoy Pilligundla, and Hadi Esmaeilzadeh. Chameleon: Adaptive code optimization for expedited deep neural network compilation. In *ICLR*, 2020.

[118] Byung Hoon Ahn, Jinwon Lee, Jamie Menjay Lin, Hsin-Pai Cheng, Jilei Hou, and Hadi Esmaeilzadeh. Ordering chaos: Memory-aware scheduling of irregularly wired neural networks for edge devices. In *MLSys*, 2020.

[119] Sungju Ryu, Hyungjun Kim, Wooseok Yi, and Jae-Joon Kim. Bitblade: Area and energy-efficient precision-scalable neural network accelerator with bitwise summation. In *DAC*, 2019.

[120] Amir Yazdanbakhsh, Michael Brzozowski, Behnam Khaleghi, Soroush Ghodrati, Kambiz Samadi, Nam Sung Kim, and Hadi Esmaeilzadeh. Flexigan: An end-to-end solution for fpga acceleration of generative adversarial networks. In *FCCM*, 2018.

[121] Bita Darvish Rouhani, Mohammad Samragh, Mojan Javaheripi, Tara Javidi, and Farinaz Koushanfar. Deepfense: Online accelerated defense against adversarial deep learning. In *ICCAD*, 2018.

[122] Mohammad Samragh, Mojan Javaheripi, and Farinaz Koushanfar. Encodeep: Realizing bit-flexible encoding for deep neural networks. *TECS*, 2019.

[123] Shehzeen Hussain, Mojan Javaheripi, Paarth Neekhara, Ryan Kastner, and Farinaz Koushanfar. Fastwave: Accelerating autoregressive convolutional neural networks on fpga. *arXiv*, 2020.

[124] Charles Eckert, Xiaowei Wang, Jingcheng Wang, Arun Subramaniyan, Ravi Iyer, Dennis Sylvester, David Blaaauw, and Reetuparna Das. Neural cache: Bit-serial in-cache acceleration of deep neural networks. In *ISCA*, 2018.