Enabling Fine-Grained Spatial Multitasking on Systolic-Array NPUs Using Dataflow Mirroring

Jinwoo Choi, *Student Member, IEEE*, Yeonan Ha, *Student Member, IEEE*, Jounghoo Lee, *Student Member, IEEE*, Sangsu Lee, *Student Member, IEEE*, Jinho Lee, *Member, IEEE*, Hanhwi Jang, *Member, IEEE*, and Youngsok Kim, *Member, IEEE*

***Abstract*—Neural Processing Units (NPUs) frequently suffer from low hardware utilization as the efficiency of their systolic arrays heavily depends on the characteristics of a deep neural network (DNN). Spatial multitasking is a promising solution to overcome the low NPU hardware utilization; however, the state-of-the-art spatial-multitasking NPU architecture achieves sub-optimal performance due to its coarse-grained systolic-array distribution and incurs significant implementation costs. In this paper, we propose *dataflow-mirroring NPU (DM-NPU)*, a novel spatial-multitasking NPU architecture supporting fine-grained systolic-array distribution. The key idea of DM-NPU is to reverse the dataflows of co-located DNNs in horizontal and/or vertical directions. DM-NPU can place allocation boundaries between any adjacent processing elements of a systolic array, both horizontally and vertically. We then propose *DM-Perf*, an accurate systolic-array NPU performance model, to maximize the spatial-multitasking performance of DM-NPU. Utilizing the existing performance models achieves sub-optimal performance as they cannot accurately capture the resource contention caused by spatial multitasking. DM-Perf, on the other hand, exploits the per-layer performance profiles of a DNN to accurately capture the resource contention. Our evaluation using MLPerf DNNs shows that DM-NPU and DM-Perf can greatly improve the performance by up to 35.1% over the state-of-the-art NPU architecture and performance model.**

***Index Terms*—Neural Processing Unit (NPU), systolic array, spatial multitasking, performance modeling.**

Manuscript received 31 October 2022; revised 14 July 2023; accepted 15 July 2023. Date of publication 1 August 2023; date of current version 8 November 2023. This work was supported in part by the National Research Foundation of Korea (NRF) under Grants 2022R1C1C1008131, 2021R1F1A1062902, and 2022R1C1C1011307, and in part by the Institute of Information and Communications Technology Planning and Evaluation (IITP) under Grant 2020-0-01361 funded by the Korean government (MSIT), in part by the Yonsei Signature Research Cluster Program under Grant 2022-22-0002 funded by Yonsei University, and in part by the Samsung Advanced Institute of Technology (SAIT). Recommended for acceptance by X. Fu. *(Corresponding authors: Youngsok Kim; Hanhwi Jang.)*

Jinwoo Choi, Yeonan Ha, Jounghoo Lee, Sangsu Lee, and Youngsok Kim are with the Department of Computer Science, Yonsei University, Seoul 03722, South Korea (e-mail: jinwoo1029@yonsei.ac.kr; yeonan@yonsei.ac.kr; jounghoolee@yonsei.ac.kr; sangtongsu@yonsei.ac.kr; youngsok@yonsei.ac.kr).

Jinho Lee is with the Department of Electrical and Computer Engineering, Seoul National University, Seoul 08826, South Korea (e-mail: leejinho@snu.ac.kr).

Hanhwi Jang is with the Department of Electrical and Computer Engineering, Ajou University, Suwon 16499, South Korea (e-mail: hanhwi@ajou.ac.kr).

Digital Object Identifier 10.1109/TC.2023.3299030

I. Introduction

Deep Neural Networks (DNNs) have become increasingly popular as they can make highly accurate predictions for various tasks such as image classification [1], [2], [3], translation [4], and personalized recommendation [5]. A DNN makes a prediction by applying a series of layers on its input data. A layer takes as input a set of input activations, performs some operations on the input activations, and produces output activations. The output activations are then used as the input activations for subsequent layers of the DNN. Different types of layers perform different operations on their input activations. For example, convolutional layers, known for the major computation in DNNs, produce higher-level abstraction of the input activations using per-output-channel filters, whereas pooling layers down-sample the given input activations by applying a fixed function to spatially-adjacent input activations (e.g., average, max) [6].

To efficiently process the layers of a DNN, various Neural Processing Units (NPUs) have been proposed. Although a variety of NPU architectures exist, employing a *systolic array* [7], a two-dimensional grid of tightly-coupled Processing Elements (PEs), has been a popular design choice due to its high throughput and energy efficiency [6]. Systolic arrays enable highly efficient execution of matrix multiplication, a key operation of convolutional and fully-connected layers in DNNs, by transferring data between spatially-adjacent PEs and performing a large number of multiply-accumulate (MAC) operations in parallel. Examples of the NPUs utilizing a systolic array include Google's Tensor Processing Units (TPUs) [8], [9] and Samsung's mobile NPU [10].

Aimed at accelerating single-DNN executions, most of the existing NPUs have supported executing only a single DNN at a time. Unfortunately, the single-NN execution model makes NPUs achieve low hardware utilization due to a mismatch between an NPU's systolic array configuration and the layers of a DNN [11]. Aimed at increasing the hardware utilization, prior studies propose to implement *spatial multitasking* which partitions and distributes an NPU's hardware resources to co-located DNNs, allowing the NPU to concurrently execute the DNNs at the same time [12]. By concurrently executing multiple DNNs, spatial multitasking can greatly improve not only the hardware utilization, but also multitasking performance such as system throughput and average normalized turnaround time [13].

To maximize the hardware utilization and multitasking performance, it is desirable for a spatial-multitasking NPU architecture to distribute its abundant hardware resources to DNNs in a fine-grained manner without incurring large hardware implementation costs. In this sense, Planaria [12], the state-of-the-art spatial-multitasking NPU architecture, suffers from its coarse-grained systolic-array distribution granularity and excessive hardware implementation costs. Planaria implements spatial multitasking by partitioning its systolic arrays into fixed-size sub-arrays (e.g., 128 × 128 PEs into 16 32 × 32 PEs) and by distributing the sub-arrays to DNNs. The sub-array-based design enables diverse systolic-array distribution patterns; however, its systolic-array allocation granularity is bounded by the sub-array size and requires an expensive all-to-all high-radix crossbar for connecting all the sub-arrays. To achieve fine-grained systolic-array allocation, Planaria may use a smaller sub-array size (e.g., 1 × 1 PE). Unfortunately, using a small sub-array size incurs excessive hardware implementation costs due to the high-radix crossbar, and thus Planaria cannot achieve efficient fine-grained spatial multitasking.

In this paper, we present *dataflow-mirroring NPU (DM-NPU)*, a novel systolic-array NPU architecture which realizes highly efficient fine-grained spatial multitasking with low hardware implementation costs. The key idea of DM-NPU is to reverse the horizontal and/or the vertical dataflows of co-located DNNs. Reversing the dataflows allows a systolic-array allocation boundary to be set between any adjacent PE rows and columns, and thus enables fine-grained systolic-array allocation. First, DM-NPU prevents any interference between the DNNs by making their data flow in reverse directions from an allocation boundary to the borders of the systolic array. Second, to achieve the fine-grained systolic-array allocation granularity with low implementation costs, DM-NPU employs an omni-directional inter-PE network which allows the boundaries between any adjacent PE rows and columns to be an allocation boundary. Third, DM-NPU supports up to four-way allocation by exploiting the rectangular shape of the systolic array which allows up to three allocation boundaries while ensuring that the allocated systolic array regions remain rectangular.

Then, to achieve high spatial-multitasking performance with DM-NPU, we propose DM-Perf, an accurate systolic-array NPU performance model which captures the contention on the NPU hardware resources shared by co-located DNNs. Accurate modeling of the shared resources is essential for achieving high multitasking performance as the performance heavily relies on how a systolic array is allocated to the DNNs. We find that the existing systolic-array NPU performance models do not consider the contention, and thus produce sub-optimal systolic-array allocations achieving low multitasking performance. DM-Perf, on the other hand, produces a high-performance systolic-array allocation for a given set of co-located DNNs by accurately capturing the contention using the per-layer memory bandwidth consumptions of the DNNs.

To demonstrate the high effectiveness of DM-NPU and DM-Perf, we prototype DM-Perf on top of Google's Tensor Processing Unit (TPU) [8] and evaluate its performance using diverse 2- and 4-way spatial-multitasking scenarios involving nine MLPerf DNNs. Our evaluation shows that DM-NPU can improve the multitasking performance by up to 73.1% over the baseline TPU. The evaluation also shows that DM-Perf can generate systolic-array allocations achieving 35.1% higher multitasking performance compared to the existing NPU performance model. Compared to the baseline TPU employing the state-of-the-art NPU performance model, performing spatial multitasking on DM-NPU with DM-Perf achieves a geometric mean throughput improvement of 18.98%. The results clearly demonstrate that DM-NPU and DM-Perf can greatly improve the performance.

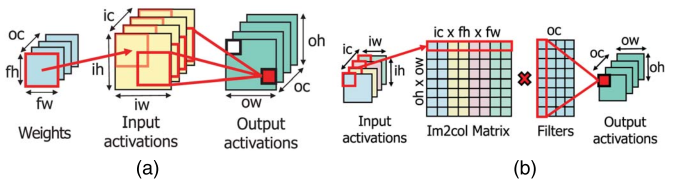

Fig. 1. The operations of a convolutional layer and an image-to-column-based execution of a convolutional layer. (a) Convolutional layer. (b) Image-to-column convolution.

In summary, we make the following key contributions:

- We propose DM-NPU, a systolic-array NPU architecture which enables fine-grained spatial multitasking by allocating its systolic array to DNNs at the granularity of single PE rows and columns. DM-NPU reverses the dataflows of the DNNs so that an allocation boundary can be set between any adjacent PE rows and columns.
- We present DM-Perf, an accurate NPU performance model which exploits the per-layer memory bandwidth consumption of a DNN to capture the contention on the shared NPU hardware resources. DM-Perf utilizes the per-layer memory bandwidth profiles of co-located DNNs to accurately estimate the DNNs' slowdowns due to fine-grained spatial multitasking on DM-NPU.
- We show the high effectiveness of DM-NPU and DM-Perf using diverse spatial-multitasking scenarios using MLPerf DNNs. Our evaluation shows that DM-NPU improve the multitasking performance by up to 26.1% over the state-of-the-art coarse-grained systolic-array NPU architecture, and that DM-Perf produces optimal systolic-array allocations achieving 35.1% higher performance over the prior NPU performance models.

II. Background

A. Deep Neural Networks

A Deep Neural Network (DNN) makes a prediction by applying a series of layers on input data. For example, Convolutional Neural Networks (CNNs), one of the most widely-used types of DNNs, consist of convolutional and fully-connected layers, along with pooling, normalization, and activation layers [6]. The majority of a CNN's computation comes from convolutional and fully connected layers. Fig. 1(a) depicts the operation of a convolutional layer. A convolutional layer takes as input

$ih \times iw \times ic$ input activations where ih, iw, ic are the input height, width, and channel count, respectively. Then, it slides a $fh \times fw$ filter, where fh is the filter height and fw is the filter width, over the input activations and computes the weighted sum of the overlapped input activations and the filter weights, producing a single channel of $oh \times ow$ output activations where oh and ow are the output height and width, respectively. A convolutional layer has oc filters of the same size, so the layer produces a total of oc output channels. The operations of a convolutional layer can be summarized as:

$$O[h][w][c] = \sum_{i=1}^{fh}\sum_{j=1}^{fw}\sum_{k=1}^{ic} I[h + i \times sh][w + j \times sw][k] \times F[i][j][k][c] \quad (1)$$

where I, O, and F are the input activations, output activations, and filter weights, respectively. sh is the stride along height, and sw is the stride along width.

A convolutional layer is typically implemented as a matrix multiplication using the image-to-column (im2col) transformation [6]. As depicted in Fig. 1(b), the im2col-based execution of a convolutional layer flattens the input activations and filter weights into two-dimensional matrices. When flattening the input activations, the im2col transformation takes into account not only the input activations, filter, and output activation sizes, but also the strides and paddings. Accordingly, the im2col transformation produces a flattened input activation matrix having $oh \times ow$ rows and $ic \times fh \times fw$ columns using the input activations, and multiplies the input activation matrix with a flattened filter weight matrix which consists of $ic \times fh \times fw$ rows and oc columns. The matrix multiplication then produces a result matrix with $oh \times ow$ rows and oc columns whose elements are the output activations of the layer. A fully-connected layer can be thought of as a convolutional layer whose filter size is identical to the input activation size.

B. Systolic-Array NPU Architectures

To accommodate the ever-increasing computational demand of DNNs, Neural Processing Units (NPUs), the hardware accelerators tailored for accelerating the operations of DNNs [8], [14]. Many NPUs exploit a systolic array of Processing Elements (PEs) to accelerate matrix multiplication which is a key operation of the im2col-based convolutional layer implementations [6]. Each of the PEs can process one multiply-accumulate operation per cycle and can send/receive data to/from its adjacent PEs.

How a systolic array performs a matrix multiplication depends on its dataflow, which defines how a DNN layer's data flow between the PEs. Fig. 2(a) depicts how the PEs perform fast matrix multiplication using the Weight Stationary (WS) dataflow by associating one filter weight to a PE and making the filter weights remain stationary on their PEs [6]. The weights of a filter get distributed to the PEs belonging to the same PE column, making different PE columns process different output channels. Then, the flattened input activations get streamed horizontally across the PEs. At each cycle, each PE multiplies

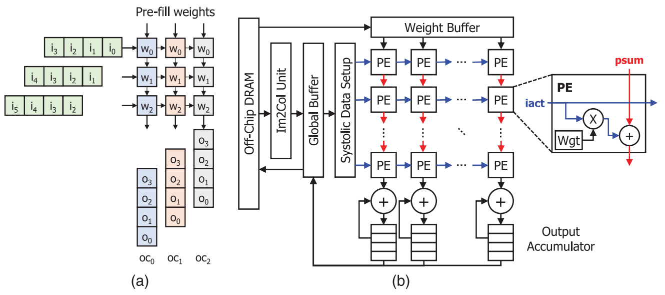

Fig. 2. Working model of the Weight Stationary (WS) dataflow and our baseline systolic-array NPU architecture implementing the WS dataflow. (a) WS dataflow. (b) Systolic-array NPU architecture.

its filter weight with the input activation received from the PE on the left, accumulates the multiplication result with the partial sum of the output activation received from the PE above, and streams the updated partial sum to the PE below. Once a partial sum reaches a bottom-most PE, it gets accumulated to the corresponding output accumulator queue (ACCQ) entry located below the systolic array.

Fig. 2(b) shows our baseline systolic-array NPU architecture modeled after Google's Tensor Processing Unit (TPU) implementing the WS dataflow. The baseline NPU architecture is equipped with an off-chip DRAM, an im2col unit, a Global Buffer (GB), a Systolic Data Setup (SDS) unit, a systolic array, and a set of ACCQs. The DRAM stores all the input activations, output activations, and filter weights of a DNN's layers. The baseline NPU architecture executes a convolutional layer as follows. First, it loads the layer's input activations from the DRAM, performs the im2col transformation on the input activations using the im2col unit, and stores the flattened input activations in the GB. The filter weights for the tile get loaded to the weight buffer (WB). Then, it performs the matrix multiplication using the PEs by pre-filling the filter weights stored in the WB to the PEs, followed by feeding the input activations stored in the GB to the left-most PEs with the SDS unit according to the WS dataflow. After all the partial sums get accumulated to their corresponding ACCQ entries, the ACCQ entries get transferred to the GB. The output activations stored in the GB can be used by the next layer being executed by the baseline NPU architecture. Otherwise, the output activations get stored to the DRAM.

As the numbers of available PE rows/columns and ACCQ entries are limited, the baseline NPU architecture often employ tiling to fully execute a layer (e.g., a layer's output channel count is larger than the number of PE columns). When tiling gets applied to, the baseline NPU architecture produces one tile (portion) of a layer's output activations at a time, rather than producing all the output activations at once. The size of the tiles is typically set to produce (the number of ACCQ entries per PE column) × (the PE column count) output activations per tile, and the flattened input activations and filter weights get tiled accordingly. Due to the tiling, the amount of GB required for storing the input activations, filter weights, and output activations of a tile frequently becomes much smaller than the GB capacity. The baseline NPU architecture utilizes the remaining GB capacity to prefetch the input activations and filter weights of the next tiles to be processed [15]. Prefetching the next tiles' inputs allows the baseline NPU architecture to hide the long DRAM-to-GB data transfer latency.

C. Systolic-Array NPU Performance Models

When it comes to maximizing the performance and hardware utilization of an NPU, being able to accurately predict the execution latency of a DNN running on the NPU is highly beneficial. Aimed at accurately estimating the execution time of a DNN running on a systolic-array NPU with low computational overhead, prior studies have proposed various systolic-array NPU performance models [16], [17], [18]. Samajdar et al. [16] propose an analytical model to accelerate systolic-array NPU design space explorations. The analytical model predicts a DNN's execution time using the input activation, filter weight, and stride sizes and takes into account the dataflow of the target NPU. Baek et al. [18] classify the execution time of a DNN into compute and memory-fetch cycles, and propose a performance model to estimate the two cycles for convolutional and fully-connected layers. Then, they utilize the estimated compute and memory-fetch cycles to improve the temporal-multitasking performance on systolic-array NPUs. Choi and Rhu [17] propose PREMA, a performance model designed to estimate the execution time of a DNN running on the systolic-array NPUs implementing the WS dataflow, and utilize PREMA to implement a predictive scheduling algorithm for preemptive systolic-array NPUs. Among the NPU performance models, we focus on PREMA as it has been validated against Google's TPU and is the state-of-the-art systolic-array NPU performance model.

PREMA [17] employs a systolic array-based NPU performance model which predicts the execution time of a DNN by assuming that all the NPU hardware resources (e.g., PEs, GB size) are allocated to the DNN at a time. The performance model predicts the per-layer execution times of a DNN, and then aggregates the per-layer execution times to derive the end-to-end execution time of the DNN. To take into account the effects of layer tiling, the performance model first computes the size and the number of tiles of a layer. Then, the performance model calculates the tile's $MemLat$, the time it takes to load the tile's input activations and filter weights from the off-chip DRAM to the GB, and $ComLat$, the time it takes for the PEs of the NPU to process the operations of the tile using the data stored in the GB. The performance model treats the last tile of the layer, which is typically smaller than the others due to a mismatch between the output activation size and the PE and ACCQ sizes, as a special case and computes the last tile's $MemLat$ and $ComLat$ separately. After that, the performance model derives the execution time of the layer as:

$$\begin{aligned} &\max\left(ComLat_{FirstTile}, MemLat_{FirstTile}\right) \times (\#tiles - 1) \\ &\quad + \max\left(ComLat_{LastTile}, MemLat_{LastTile}\right). \end{aligned} \tag{2}$$

Note that the performance model assumes a static off-chip DRAM bandwidth for simplicity and takes into account the effects of prefetching the next tiles' input activations and filter weights from the DRAM to the remaining GB capacity.

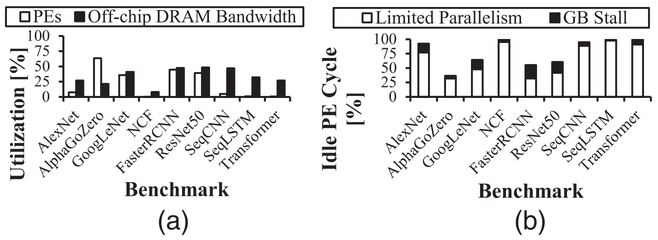

Fig. 3. Low NPU hardware resource utilizations of single DNN inference executions with a batch size of four on Google's TPU [8]. (a) Hardware resource utilizations. (b) Idle PE cycle breakdown.

III. Motivation and Design Goals

A. Underutilized NPU Hardware Resources

Aimed at accelerating single-DNN execution, most of the existing systolic-array NPU architectures execute only one DNN at a time by allocating all of its hardware resources to the DNN. Unfortunately, we find that blindly allocating all the NPU hardware resources to a single DNN incurs low hardware resource utilization and multitasking performance. Fig. 3(a) shows the PE and off-chip DRAM bandwidth utilizations of Google's TPU when it executes a single MLPerf DNN inference at a time with a batch size of four. The results show that, on average, only 22.0% of the PEs and 33.4% of the off-chip DRAM bandwidth are utilized.

To gain further insights into the low NPU hardware utilizations, we classify the idle PE cycles of the single-DNN executions into two types: limited parallelism cycles and GB stall cycles. First, we classify an idle PE cycle as a limited parallelism cycle if the operations of a layer/tile cannot fully utilize all the available PEs. For example, if a layer/tile produces fewer output channels than the available PE column count, then some PE columns will remain idle due to the insufficient output channel count. Second, we treat an idle PE cycle as a GB stall cycle when the PEs cannot perform matrix multiplications as the input activations and/or filter weights have not been fully loaded from the off-chip DRAM to the GB and WB. An idle PE cycle always gets identified as either limited parallelism or GB stall, but not both. The limited parallelism cycles reflect the idle PE cycles during the computation, whereas the GB stall cycles account for the DRAM access latency before the computation begins.

Fig. 3(b) shows the contributions of the two types of the idle PE cycles when executing a single DNN inference with a batch size of four on the TPU. The majority of the idle PE cycles are the limited parallelism cycles. This indicates that employing spatial multitasking, which distributes the NPU hardware resources to multiple DNNs and concurrently executes the DNNs at the same time, has high potential for greatly improving the NPU hardware resource utilization. By distributing the PEs to co-located DNNs, the idle PE cycles would decrease as the number of PEs each DNN needs to populate decreases. For example, distributing the PE columns to DNNs reduces the output channel count required for a DNN's layer to fully utilize the allocated PE columns. As a result, spatial multitasking can reduce the limited parallelism cycles and better

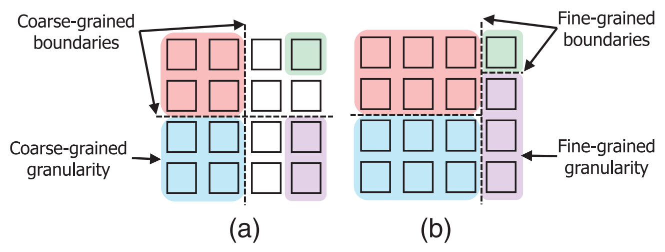

Fig. 4. Coarse- and fine-grained systolic array allocations. (a) Coarse-grained. (b) Fine-grained (Ours).

utilize the hardware resources available on systolic-array NPU architectures.

In order to improve the NPU hardware resource utilization, Planaria [12], a systolic-array NPU architecture implementing coarse-grained spatial multitasking, has been proposed. Planaria, as shown in Fig. 4(a), partitions a large systolic array into a set of fixed-sized sub-arrays (e.g., 128 $\times$ 128 PEs into 16 32 $\times$ 32 PEs) and distributes the sub-arrays to co-located DNNs. Then, Planaria concurrently executes the DNNs at the same time by making the sub-arrays execute their corresponding DNNs' layers. The sub-arrays are connected through an all-to-all high-radix crossbar, so Planaria can still allocate all of its hardware resources to a single DNN in case of single-DNN executions.

Despite the improvements in NPU hardware resource utilization and performance, Planaria's *coarse-grained* spatial multitasking still suffers from low NPU hardware resource utilization caused by the limited parallelism cycles. As the PEs can be distributed to co-located DNNs at the granularity of the fixed-sized sub-arrays, Planaria can still incur a notable number of the limited parallelism cycles as a DNN's layers often do not have sufficiently large amounts of computation. To minimize the limited parallelism cycles, Planaria may choose to employ 1 $\times$ 1 PE as its sub-array size. However, selecting such a small sub-array size would incur intolerable hardware implementation costs as all the PEs need to be connected through a single all-to-all high-radix crossbar. For example, Planaria requires an all-to-all high-radix crossbar which connects all the 16,384 PEs when employing 1 $\times$ 1-PE sub-arrays on a 128 $\times$ 128-PE systolic array.

To overcome the limited improvements in NPU hardware resource utilization and performance, we claim that systolic-array NPUs should support efficient *fine-grained* spatial multitasking which allows systolic-array allocation boundaries to be set between any adjacent PE rows and columns. The fine-grained spatial multitasking offers high flexibility which can effectively reduce the limited parallelism cycles. It can produce heterogeneous sub-arrays having different PE row and column counts, and can easily adapt to diverse layer and tile patterns of DNNs. Planaria, on the other hand, gets bounded by its fixed-sized sub-arrays, making it difficult and costly to provide similar flexibility in terms of allocating the PEs of a systolic array to DNNs. Fig. 4(b) illustrates an example fine-grained systolic-array allocation. In the example, there are three systolic-array allocation boundaries. The first boundary is set vertically between the third and fourth PE columns, and then the second and third boundaries are set between two pairs of PE rows. The three boundaries collectively generate four sub-arrays, which have different PE rows and columns, and four DNNs can run in parallel using the four sub-arrays.

To support fine-grained spatial multitasking, we need an efficient method to provide input activation and filter weights to the sub-arrays with minimal hardware costs. As any PE could be on the boundaries of allocated sub-arrays, a naïve extension of systolic-array NPU architectures, where each sub-array receives its input activations and filter weights from its left- and top-most PEs, respectively, requires every PE to be directly connected to the SDS unit, WB, and ACCQs. Unfortunately, the naïve implementation puts additional wires from all PEs to the units and buffers surrounding the systolic array, incurring unacceptable chip area and power/energy consumption overheads. Thus, we need a new efficient way to flow data to sub-arrays and derive our first design goal as follows.

- **Design Goal #1:** Enable highly flexible systolic-array allocations by implementing fine-grained spatial multitasking on top of the existing systolic-array NPU architectures with low hardware implementation costs.

B. Lack of Shared NPU Hardware Resource Modeling

Although a fine-grained spatial-multitasking NPU architecture becomes readily available, an important problem of how to allocate the NPU hardware resources to co-located DNNs remains. As DNNs have different characteristics and NPU hardware resource usage patterns, identifying the optimal NPU hardware resource allocation is crucial to maximize the performance benefits of spatial multitasking.

A naïve approach for identifying the optimal NPU hardware resource allocation for a DNN multitasking workload would be to execute and profile all possible resource allocation for the workload; however, the naïve approach is infeasible due to the long DNN execution times and the huge resource allocation space. Alternatively, systolic-array NPU performance models allow us to quickly estimate the execution time of a DNN and predict the optimal allocation for the workload using the estimated DNN execution times. Unfortunately, we find that employing the existing performance models to identify the optimal allocation for fine-grained spatial multitasking results in low performance.

To demonstrate the difficulty of finding the optimal allocation using the existing systolic-array NPU performance models, we collect and compare the system throughput (STP) achieved by (1) all the possible fine-grained systolic-array allocations, and (2) the allocation estimated to achieve the highest STP by PREMA [17], the state-of-the-art systolic-array NPU performance model, for nine spatial-multitasking workloads on a hypothetical spatial-multitasking-enabled TPU. We assume that the hypothetical TPU faithfully implements the aforementioned fine-grained systolic-array allocation which allows systolic-array allocation boundaries to be set between any adjacent PE rows and columns. The hypothetical TPU evenly distributes the GB capacity and off-chip DRAM bandwidth to the DNNs. The

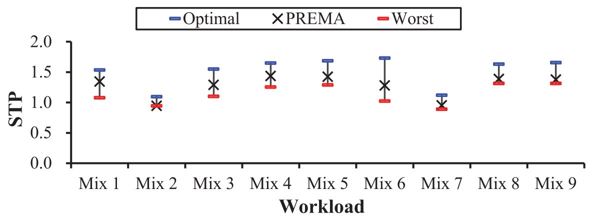

Fig. 5. Employing PREMA [17] results in sub-optimal NPU hardware resource allocations for various DNN multitasking workloads.

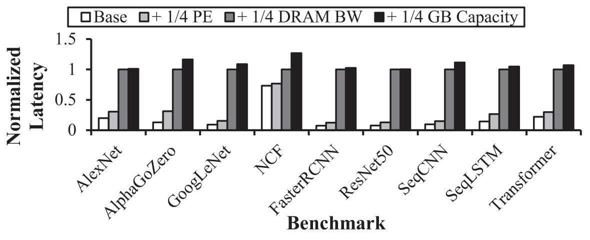

Fig. 6. Impact of spatial multitasking on MLPerf DNN execution latencies.

workloads consist of MLPerf DNNs (see Table III for details), and the DNNs are set to have a batch size of four.

Fig. 5 shows that the best-performing allocation predicted by PREMA achieves much lower STP than that of the optimal allocation, and thus the existing performance models cannot readily be used for estimating the optimal allocation. Our analysis reveals that employing the existing performance models results in sub-optimal allocations as they do not take into account the contention on the NPU hardware resources shared between co-located DNNs (e.g., GB capacity, off-chip DRAM bandwidth). The performance models target single-DNN executions and/or temporal multitasking, and thus blindly assume that all the NPU hardware resources get allocated to a single DNN. However, the NPU hardware resources get distributed to the DNNs with spatial multitasking, making co-located DNNs get assigned subsets of the NPU hardware resources (e.g., fewer PEs, smaller GB sizes) and incur higher execution times. In addition, the performance models assume a static off-chip DRAM bandwidth for simplicity, but concurrently-executing DNNs compete for the limited amount of the DRAM bandwidth and experience higher memory-access latencies due to the contention on the DRAM bandwidth.

Aimed at verifying our analysis, we examine the impact of the NPU hardware resource sharing on DNN execution latency. For the purpose, we mimic a four-way spatial multitasking scenario by scaling down the baseline systolic-array NPU architecture's hardware resources which get shared between DNNs with spatial multitasking (i.e., PEs, off-chip DRAM bandwidth, and GB capacity) by a quarter. Fig. 6 shows how the execution latency of nine MLPerf DNNs changes as we scale down the shared NPU hardware resources. When we reduce the number of PEs to a quarter, the DNN executions become 1.61$\times$ slower on average. Furthermore, scaling down the off-chip DRAM bandwidth and GB capacity incurs in 4.26$\times$ and 1.09$\times$ additional execution latencies, respectively. The results clearly indicate that the contention on the shared NPU hardware resources has significant impact on the DNN execution latencies. Thus, to accurately identify the optimal NPU hardware resource allocation, we need a new systolic-array NPU performance model which captures the contention on the shared NPU hardware resources. This leads to our second design goal:

- **Design Goal #2:** Develop an accurate contention-aware systolic-array NPU performance model for fine-grained spatial multitasking which captures the contention on the hardware resources shared between co-located DNNs.

IV. Dataflow Mirroring

This section presents *dataflow mirroring*, an efficient fine-grained systolic-array allocation mechanism designed to realize fine-grained spatial multitasking on systolic-array NPU architectures with low hardware implementation costs. We also develop dataflow-mirroring performance model (DM-Perf) which accurately captures the contention on the shared NPU hardware resources. After that, we describe dataflow-mirroring scheduler (DM-Scheduler) which serves as a software runtime for DM-NPU and achieves high spatial-multitasking performance using DM-Perf.

A. Key Idea

The key idea of dataflow mirroring is to reverse the dataflows of co-located DNNs in horizontal and/or vertical directions. Reversing the dataflows enables fine-grained systolic array allocation and allows a two-dimensional systolic array to co-locate up to four DNNs. First, reversing the dataflows allows every row and column of a systolic array to be an allocation boundary, enabling fine-grained systolic array allocation. Second, as the dataflows can be reversed twice, each in the horizontal and vertical directions, up to three allocation boundaries can be set and up to four DNNs to be co-located on the same systolic array.

Dataflow mirroring supports two modes, input-activation mirroring (iact mirroring) and partial-sum mirroring (psum mirroring), to reverse the dataflows. The iact mirroring horizontally reverses the direction of a DNN's input activations, whereas the psum mirroring vertically reverses the direction of a DNN's partial sums with respect to a selected allocation boundary. Fig. 7 illustrates the two modes on an example 4 $\times$ 4 systolic array. The baseline WS dataflow streams the input activations colored blue in the horizontal direction from left to right and the partial sums in the vertical direction from top to bottom (Fig. 7a). When two DNNs are co-located, the systolic array distributes its PE columns or rows to the two DNNs depending on the selected mirroring mode. To distribute the PE columns, the systolic array utilizes the iact mirroring (Fig. 7b). After selecting a vertical allocation boundary, the input activations of one DNN flows left-to-right, whereas the input activations of the other DNN flows right-to-left. Once the input activations reach the allocation boundary, the systolic array simply discards them. The flow of the partial sums remains unchanged. Similarly, the psum mirroring partitions the systolic array horizontally and distributes the PE rows (Fig. 7c). The partial sums of one DNN flow downwards, whereas the partial sums of the other DNN flow upwards. The input activations of the two DNNs flow

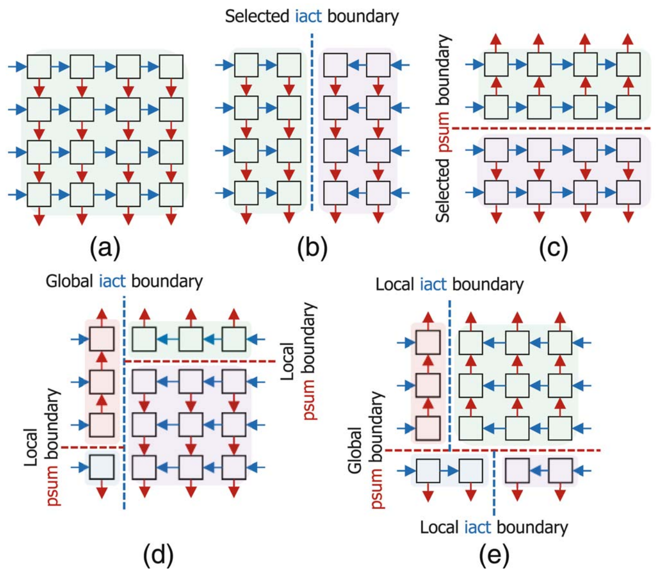

Fig. 7. Fine-grained systolic array allocation with dataflow mirroring. (a) Baseline WS. (b) Iact mirroring. (c) Psum mirroring. (d) Iact-then-psum mirroring. (e) Psum-then-iact mirroring.

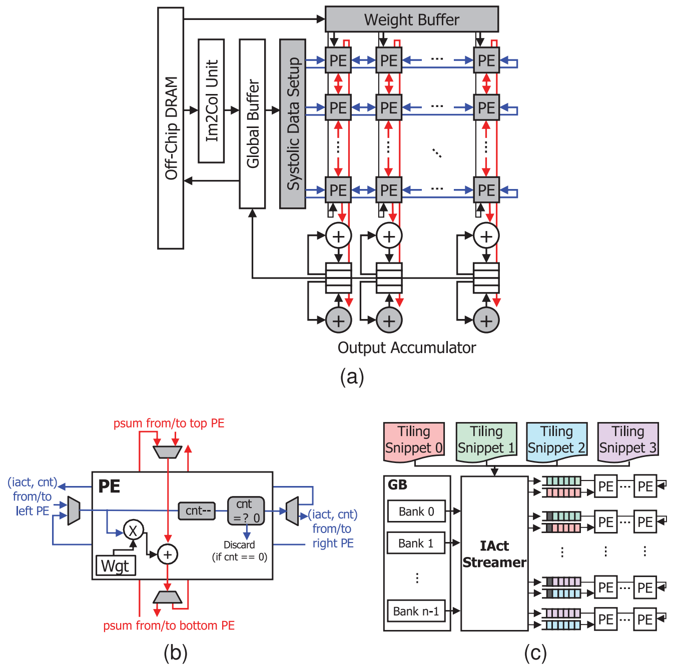

Fig. 8. Dataflow-mirroring NPU (DM-NPU) architecture built on top of Google's TPU. The extended hardware components are shaded in gray. (a) Overall architecture. (b) PE. (c) SDS unit.

left-to-right. In addition, we can apply the iact/psum mirroring on the sub-arrays partitioned by the psum/iact mirroring (Fig. 7d, e). By applying both the iact and psum mirrorings, dataflow mirroring can co-locate up to four DNNs.

B. DM-NPU: Dataflow-Mirroring NPU Architecture

We propose DM-NPU, a new systolic-array NPU architecture which implements dataflow mirroring. DM-NPU is built on top of Google's TPU [8], but the key idea, dataflow mirroring, is applicable to any type of systolic array-based architecture. Fig. 8(a) illustrates the overall architecture of DM-NPU. To support dataflow mirroring, DM-NPU extends TPU with an omni-directional inter-PE network, the lifetime counters associated with the input activations, and the SDS unit to stream the input activations of multiple DNNs.

First, DM-NPU replaces the inter-PE connections with omni-directional ones to allow the PEs to send data back and forth to the adjacent PEs. TPU's WS dataflow only needs the input activations and partial sums to flow left-to-right and top-to-bottom, respectively. However, DM-NPU's dataflow mirroring requires the data to flow in the opposite directions as well. To support the bi-directional communication, DM-NPU places additional multiplexers around each PE as shown in Fig. 8(b). The multiplexers select the appropriate data sources and destinations.

Second, DM-NPU associates a lifetime counter with its input activation to comply with the boundary set between two adjacent PE columns. The input activations of the two sub-arrays, separated by the boundary, flow toward each other, so the input activation interference can happen at the boundary. To prevent the interference, the input activations are associated with lifetime counters which specify the number of PE columns the input activation should flow through. As a PE receives an input activation, it decreases the associated counter by one. If the value of the counter reaches zero, the PE discards the input activation and does not transfer it beyond the boundary. Otherwise, the PE forwards the input activation with the counter value decreased by one to the next PE. To implement this mechanism, we put an additional subtractor to each PE. For vertically flowing data (i.e., partial sums), we do not need such additional hardware as the partial sums of the PE rows separated by dataflow mirroring (i.e., psum mirroring) flow in opposite directions, not interfering with each other.

Third, DM-NPU extends the SDS unit to feed the input activations of multiple DNNs to the systolic array. As shown in Fig. 8(c), the main part of SDS unit is the *IAct Streamer*, which transfers the input activations of up to four DNNs from the GB to the left- and right-most PEs. To decide which PEs the input activations should be fed to, the IAct Streamer takes tiling snippets of the DNNs; a DNN's tiling snippet contains the information on the memory address and size of the input activations, the tile size, and the coordinates of the allocated PEs. Then, it loads the input activations to the FIFO queues of the allocated PE rows with the corresponding lifetime counters according to the boundaries. Our design has two FIFO queues for each PE row: one for the left-most PEs and the other for the right-most PEs. For the filter weights and output activations, we add extra (pipelined) wires to the WB and ACCQs to allow the top- and bottom-most PEs.

Fourth, DM-NPU evenly distributes the GB capacity and off-chip DRAM bandwidth to co-located DNNs. DM-NPU employs a multi-banked GB, so evenly distributing the GB capacity to the DNNs is achieved by evenly allocating the GB's banks to the DNNs. To evenly distribute the DRAM bandwidth, DM-NPU issues one DRAM access request to the off-chip DRAM per cycle in turn. For example, when four DNNs get

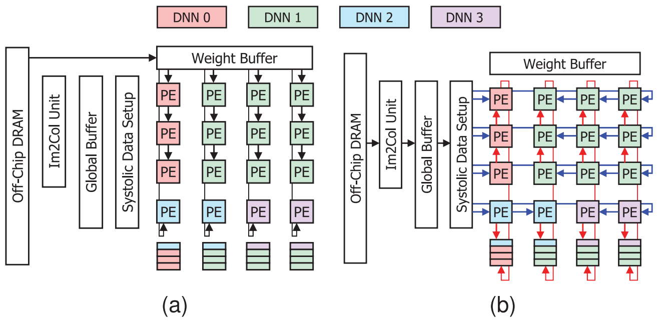

Fig. 9. Working model of DM-NPU for four-way spatial multitasking using the psum-then-iact mirroring. (a) Pre-fill weights. (b) Concurrent execution.

co-located on DM-NPU, it issues the first, second, third, and fourth DNN's DRAM access request to the DRAM in the first, second, third, and fourth cycle, respectively. When the DNN which can issue a DRAM access request does not have any outstanding request, the next DNN gets a chance to access the DRAM in that cycle.

With the proposed extensions, DM-NPU executes multiple DNNs by implementing dataflow mirroring. In the example scenario shown in Fig. 9, DM-NPU partitions its systolic array into four sub-arrays. First, DM-NPU pre-fills the weights of four co-located DNNs to their corresponding PEs by exploiting the newly-added wires from the WB to the systolic array. Then, the extended SDS unit brings the input activations of the DNNs from the GB to the PEs with the corresponding lifetime counter values. After that, DM-NPU concurrently executes the DNNs and produce partial sums using all the PEs. The partial sums, in turn, concurrently populate the corresponding ACCQ entries using the extended wires from the systolic array to the ACCQs.

C. *DM-Perf: Dataflow-Mirroring Performance Model*

We now propose DM-Perf, a new systolic-array NPU performance model for DM-NPU to find the optimal NPU hardware resource allocation. As DNNs share the hardware resources on DM-NPU, it is important to consider how the sharing affects the execution times of the DNNs for accurate spatial-multitasking performance modeling. To accurately estimate the DNN execution times, DM-Perf considers three performance-critical shared NPU hardware resources: the systolic array, GB, and off-chip DRAM bandwidth.

Algorithm 1 details DM-Perf. DM-Perf estimates the execution time of the tiles which form a DNN layer (line 12). Then, it accumulates the execution times of all the tiles of the DNN's layers to compute the DNN's execution time. The execution time of a tile consists of (1) the computation latency spent by performing computation on an allocated sub-array, and (2) the memory fetch latency to load the input data for a tile from the off-chip DRAM to the GB. The numbers of allocated PEs and ACCQ entries affect the size of a layer's tile so that a tile fits within the NPU hardware resources allocated to the layer. For a layer having an $N \times K$ input activation matrix and a $K \times M$ filter weight matrix, DM-Perf computes the sizes of the tiled input activation and filter weight matrices, $n \times k$ and $k \times m$, respectively, according the allocated sub-array size. n, k, m of

Algorithm 1 DM-Perf

```
1:  $GB$: Allocated GB capacity
2:  $BW_{DRAM}$: off-chip DRAM Bandwidth
3:  $Util_{DRAM}$: Utilization of DRAM of layer
4:  $M, K, N$: Im2col data size of layer
5:  $m, k, n$: Tiled im2col data of tile
6:  $C_{latency}$: Compute latency of tile
7:  $M_{latency}$: Memory fetch latency of tile
8:  $end_{iact}, start_{iact}$: End, start address of input activation of tile
9:  $Time_{estimated}$: Total execution latency of a DNN
10: procedure DM-PERF.ESTIMATEEXECTIME
11:     $Time_{estimated} = 0$
12:     for Tiles in Layers do
13:         $GB_{Leftover} = GB$
14:         $Data_{Layer} = M \times K + K \times N$
15:         for each $(m,k,n)$ in Tiles do
16:             $C_{latency} = m + 2 \times k + n$
17:             $iact_{size} = end_{iact} - start_{iact}$
18:             $M_{latency} = (iact_{size} + k \times m)/(BW_{DRAM} \times Util_{DRAM})$
19:             if $GB_{Leftover} \le 0$ then
20:                 $Time_{estimated} += (C_{latency} + M_{latency})$
21:             else if $(GB_{Leftover} > 0)$ and $(Data_{Layer} < GB)$ then
22:                 $Time_{estimated} += C_{latency}$
23:             else
24:                 $Time_{estimated} += \max(C_{latency}, M_{latency})$
25:                 $GB_{Leftover} -= (iact_{size} + k \times m)$
26:             end if
27:         end for
28:     end for
29: return $Time_{estimated}$
30: end procedure
```

each tile can become as large as the number of ACCQ entries per PE column, the height of the sub-array, and the width of the sub-array, respectively. Line 16 of Algorithm 1 shows the formula for calculating the compute latency of a tile using the tile's size. When a tile of $m \times k \times n$ is given, filling the input activations and filter weights into the allocated sub-array takes $n + k$ and k cycles, respectively. Then, the sub-array computes the output activations for the tile in m cycles. To accurately predict the memory fetch latency of a tile, DM-Perf considers each layer's DRAM access characteristics and the contention on the off-chip DRAM bandwidth. The memory-fetch latency is computed by dividing the amount of data loaded from the DRAM by the DRAM bandwidth. To estimate the DRAM bandwidth in the presence of contention, DM-Perf first profiles each layer's DRAM utilization a priori without executing the other DNNs. To reflect the layer's DRAM access characteristics, it uses the profiled DRAM utilization as the layer's DRAM utilization on DM-NPU if the DRAM bandwidth remains underutilized during spatial multitasking (line 18). When the DRAM bandwidth gets saturated due to spatial multitasking, we need to model the reduced per-DNN DRAM bandwidth due to the contention. As DM-NPU processes the DRAM access requests of the DNNs in a round-robin manner, DM-Perf assumes that the DRAM bandwidth gets allocated to the co-located layers of the DNNs proportional to the profiled DRAM utilizations of the layers. The following equations show how DM-Perf calculates the DRAM utilization of each layer when the layers of the DNNs execute in parallel:

$$Util_{sum} = \sum_{i=0}^{\#DNN-1} Util_{layer_i}$$
$$Util_{DRAM} = \begin{cases} Util_{layer}/Util_{sum}, & \text{if } Util_{sum} > 1 \\ & \text{(BW saturated)} \\ Util_{layer}, & \text{otherwise} \end{cases} \quad (3)$$

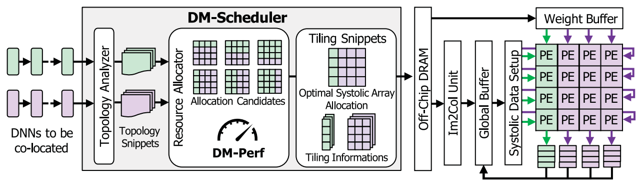

Fig. 10. Working model and components of DM-Scheduler.

Distributing the GB capacity to DNNs greatly affects the performance as the reduced per-DNN GB capacity limits the degree of prefetching the next tiles' input activations and filter weights. To model this, DM-Perf defines three cases: *non-prefetch*, *data reuse*, and *prefetch* (lines 19–26). In the first case, *non-prefetch*, the GB capacity available to a DNN is below zero, so the execution time becomes the sum of the compute latency and the memory fetch latency as the next tiles' data cannot be prefetched. In the *data reuse* case, where the GB is not full and the amount of GB needed for storing the input data for all the tiles of a layer is smaller than the allocated GB capacity, the execution latency is the same as the compute latency as all data is loaded in GB, not evicted. The execution latency, in case of *prefetch*, is determined by the larger of the compute or memory-fetch latency as the prefetching and computation happen in parallel.

D. *DM-Scheduler: Dataflow-Mirroring Scheduler*

Given a set of DNNs, DM-NPU should identify the optimal NPU hardware resource allocation to maximize the performance and hardware utilization. For this purpose, we develop *DM-Scheduler*, a runtime system for finding the optimal resource allocation. Fig. 10 depicts the working model of DM-Scheduler and its components: a topology analyzer and a resource allocator. The topology analyzer analyzes the DNNs and generates their topology snippets which contain the layer count, per-layer input activation, filter weight, and output activation sizes, etc. Then, the resource allocator estimates the execution time of each DNN on systolic-array allocation candidates using DM-Perf and the topology snippets. The allocator selects the candidate optimal for a given metric (e.g., STP) and stores the information in the tiling snippets for DM-NPU. The runtime system then distributes the hardware resources to the DNNs according to the optimal allocation recorded in the tiling snippets. When a new DNN is given or one of the co-located DNNs completes its execution, DM-Scheduler preemptively reallocates the resources by waiting for all the currently-running layers to finish their executions.

The runtime overhead from evaluating the allocation candidates might be a concern; however, it does not greatly affect the overall performance as the computation of DM-Perf would be overlapped with the execution of the already-allocated DNNs. Also, DM-NPU allows splitting the systolic array into up to four sub-arrays, limiting the search space to identifying up to three systolic-array allocation boundaries. We can further limit the search space, for example, by considering only the sub-arrays whose height and width are powers of 2. In addition, we can perform the computation offline if a set of co-located DNNs can be known in advance.

TABLE I
DM-NPU SIMULATION PARAMETERS

Parameter	Low Performance	TPU-Like	High Performance
Clock frequency	1 GHz		
Systolic array	64×64	128×128	256×256
Output accumulators	2048 entries/column		
On-chip SRAM buffer	32 banks, 32 B/cycle		
	4 MB	8 MB	16 MB
Off-chip DRAM	HBM2, 8 channels, 256 GB/s		
Computation order	Filter-major [19]		
Memory scheme	Working sets of filter and activations [19]		

V. EVALUATION

A. *Experimental Setup*

To evaluate the spatial-multitasking performance of DM-NPU, we extend SCALE-sim [16], a cycle-level systolic-array NPU architecture simulator. We augment SCALE-sim with DRAMsim3 [20] for accurate DRAM access latency modeling. To measure the energy consumption of DM-NPU, we assume a 22-nm technology and employ Accelergy [21] (for PEs and GB accesses) and DRAMsim3 [20] (for off-chip DRAM accesses). As the baseline systolic-array NPU architecture, we model Google's TPU [8] equipped with 128×128 PEs and HBM2 off-chip DRAM. Aimed at evaluating the effectiveness of DM-NPU and DM-Perf with various NPU architecture configurations, we examine not only a TPU-like configuration, but also low- and high-performance configurations having different systolic array and GB sizes. Table I shows the simulation parameters and their values.

We model Planaria [12], the state-of-the-art coarse-grained spatial-multitasking NPU architecture, by dividing the systolic array into four sub-arrays having the same height and width (i.e., four 64 × 64-PE sub-arrays). When executing co-located DNNs, the sub-arrays get allocated to the DNNs using a systolic-array NPU performance model. When there are two co-located DNNs, the performance model determines whether the sub-arrays should be distributed in row- or column-wise manner to achieve higher performance. The row- and the column-wise sub-array allocations evenly distribute the PE rows and columns to the two DNNs, respectively. When four DNNs run together on Planaria, the sub-arrays get evenly distributed to the DNNs, making each of the sub-arrays execute one of the DNNs.

As benchmarks, we select nine MLPerf DNNs [22], [23] and construct spatial-multitasking workloads with the DNNs. The DNNs use a batch size of four, and their characteristics are shown in Table II. The per-DNN PE and off-chip DRAM bandwidth utilizations are measured on the baseline TPU with the TPU-like configuration. We then classify the DNNs into four categories: memory-intensive (M), compute-intensive (C), mixed (X), and lightweight (L). After that, we construct nine DNN spatial-multitasking workloads having different co-located DNN counts and per-DNN characteristics. Table III lists the seven two-way and the two four-way spatial-multitasking workloads. For each of the spatial-multitasking

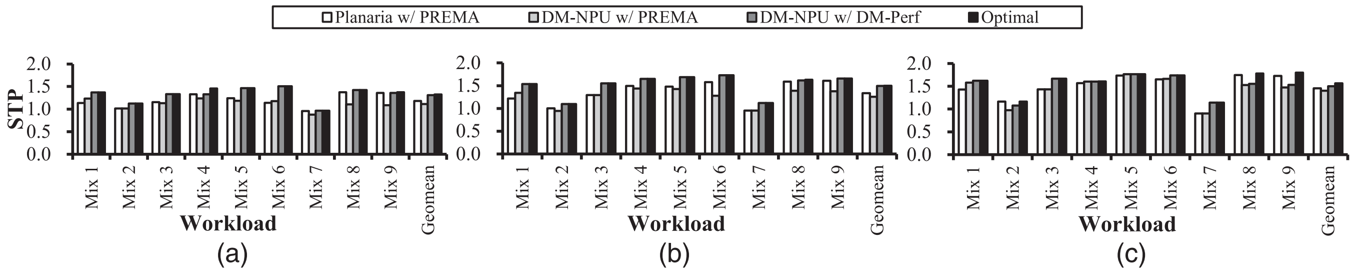

Fig. 11. STP values achieved by three combinations of NPU architecture and performance model with the nine spatial-multitasking workloads. (a) Low-performance. (b) TPU-like. (c) High-performance.

TABLE II
Characteristics of the Evaluated MLPerf DNNs

Name	PE Util.	DRAM Bandwidth Util.	Category
AlexNet	7.70%	26.94%	M
GoogleNet	35.78%	41.15%	X
ResNet-50	39.37%	48.41%	X
AlphaGoZero	63.53%	21.57%	C
NCF	0.21%	8.19%	L
FasterRCNN	44.71%	47.52%	M
seqCNN	5.06%	47.21%	M
seqLSTM	1.00%	32.30%	M
Transformer	0.79%	27.07%	M

M: memory-intensive / C: compute-intensive / X: mixed / L: lightweight

TABLE III
Evaluated Spatial-Multitasking Workloads

Workload	Benchmarks	Scenario
Mix 1	AlphaGoZero, NCF	CL
Mix 2	AlphaGoZero, SeqCNN	CM
Mix 3	NCF, FasterRCNN	LX
Mix 4	NCF, SeqLSTM	LM
Mix 5	NCF, Transformer	LM
Mix 6	NCF, AlexNet	LM
Mix 7	FasterRCNN, ResNet-50	XX
Mix 8	AlphaGoZero, ResNet-50, NCF, Transformer	CXLM
Mix 9	GoogleNet, ResNet-50, NCF, Transformer	XXLM

M: memory-intensive / C: compute-intensive / X: mixed / L: lightweight

workloads, we simulate one billion clock cycles of the underlying NPU architecture to ensure that all the co-located DNNs complete at least one iteration.

To evaluate the multitasking performance, we employ two performance metrics: system throughput (STP) and average normalized turnaround time (ANTT) [24]. STP is a system-perceived performance metric which examines the number of DNN executions completed by an NPU per unit time and is a higher-is-better metric. ANTT is a user-perceived performance metric which expresses the amount of slowdown a user experiences due to multitasking and is a lower-is-better metric. In addition, we measure the PE and off-chip DRAM bandwidth utilizations to confirm and analyze the impact of spatial multitasking on the NPU hardware resource utilization. The PE utilization hints the average number of MAC operations performed by the PEs per unit time. When PE utilization is 100%, all the PEs perform one MAC operation every cycle. Similarly, the DRAM bandwidth utilization reflects the amount of data transferred between the DRAM and GB per unit time; 100% denotes that the DRAM bandwidth is fully utilized.

B. High System Throughput

In this experiment, we evaluate the improvements in STP of DM-NPU and DM-Perf against Planaria [12] and PREMA. To perform an in-depth study on the improvements in STP, we compare three systolic-array NPU architecture and performance model combinations: Planaria with PREMA, DM-NPU with PREMA, and DM-NPU with DM-Perf. Planaria with PREMA employs Planaria as the underlying spatial-multitasking NPU architecture and PREMA as its performance model. DM-NPU with PREMA replaces the underlying spatial-multitasking NPU architecture with DM-NPU to enable fine-grained systolic-array allocation. DM-NPU with DM-Perf then replaces PREMA with DM-Perf to exploit the high modeling accuracy of DM-Perf.

Fig. 11 shows the STP values achieved by the three combinations of systolic-array NPU architecture and performance model. The figure also shows the optimal STP values which have been collected from an extensive design space exploration on the spatial-multitasking workloads and NPU architecture configurations. DM-NPU with DM-Perf achieves a geometric mean STP improvement of 10.8%, and improves STP by up to 31.9% over Planaria with PREMA. Compared to DM-NPU with PREMA, DM-NPU with DM-Perf achieves a geometric mean STP improvement of 10.8%, and up to 27.8% higher STP. The improvements show the high effectiveness of DM-NPU's fine-grained spatial multitasking and DM-Perf's accurate shared NPU hardware resource modeling. In the case of executing Mix 7 on the low-performance configuration, DM-NPU with DM-Perf achieves an STP value similar to that of the single-DNN execution. It turns out that the NPU hardware resources are already highly utilized, leaving a little room for improvement for spatial multitasking.

C. Low Average Turnaround Time

We now examine how much improvements in ANTT DM-NPU with DM-Perf can provide over the baseline Planaria with PREMA. Fig. 12 shows the ANTT values achieved by the three combinations and the STP-optimal systolic-array allocation with the spatial-multitasking workloads. The results show that DM-NPU with DM-Perf achieves geometric mean improvements in ANTT of 5.8%, 13.0%, and 5.4% with low-performance, TPU-like, and high-performance configurations, respectively, over Planaria with PREMA. Compared to DM-NPU with PREMA, DM-NPU with DM-Perf achieves up to

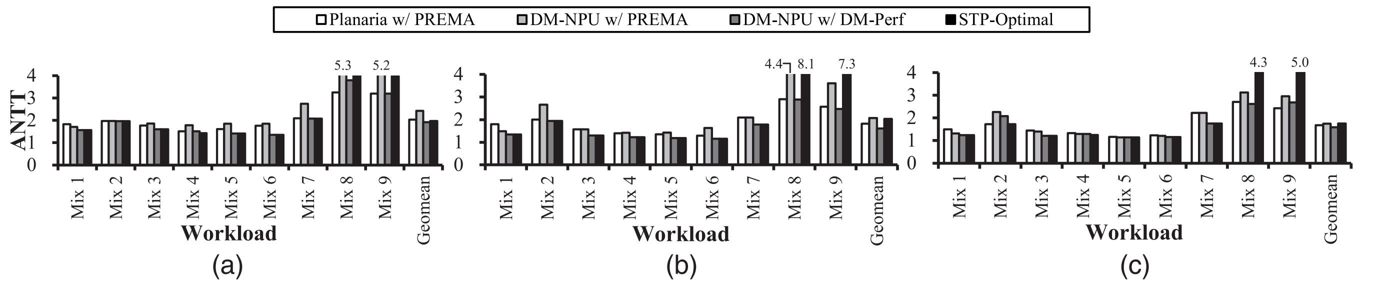

Fig. 12. ANTT values achieved by the three combinations of NPU architecture and performance model with the nine spatial-multitasking workloads. (a) Low-performance. (b) TPU-like. (c) High-performance.

53.5%, 61.3%, and 26.7% lower ANTT with low-performance, TPU-like, and high-performance configurations, respectively. Among the workloads, Mixes 8 and 9 exhibit high ANTT values as they involve four-way spatial multitasking; the amount of NPU hardware resources allocated to each of the four DNNs is much smaller than that of two-way spatial multitasking, increasing the execution times of the four DNNs. For the Mixes, DM-NPU with DM-Perf achieves lower ANTT value than that of STP-Optimal. STP-Optimal allocates the NPU hardware resources to the DNNs to achieve the highest STP, and thus its allocations do not necessarily achieve the lowest-possible ANTT.

D. High NPU Hardware Resource Utilization

Spatial multitasking improves the performance by increasing the NPU hardware resource utilization. To examine the improvements in the resource utilization, we evaluate the PE and off-chip DRAM bandwidth utilizations of PREMA and DM-NPU compared to the baseline TPU architecture.

Fig. 13(a) shows the average PE utilizations of DM-NPU and Planaria with either DM-Perf or PREMA, and their improvements over the baseline TPU with the three NPU hardware configurations. DM-NPU outperforms the baseline TPU in all the configurations, achieving up to 2.68× higher utilization with DM-Perf. The effectiveness of DM-NPU increases as the NPU scales as spatial multitasking can utilize the increased resources while a single-DNN execution often cannot. With PREMA, DM-NPU achieves slightly lower utilization than Planaria with PREMA. Our analysis reveals that this result comes from the low modeling accuracy of PREMA. Since Planaria supports coarse-grained spatial multitasking, it has fewer systolic-array allocation candidates, and thus is less sensitive to the accuracy of the performance model. DM-NPU, on the other hand, supports fine-grained spatial multitasking which is more sensitive to the modeling accuracy of the performance model.

As shown in Fig 13(b), DM-NPU with DM-Perf improves the off-chip DRAM bandwidth utilization by 41.9% (Low-perf), 48.9% (TPU-like), and 75.9% (High-perf) over the baseline TPU. The baseline TPU executes only a single DNN at a time, so it cannot fully utilize the large DRAM bandwidth. Accordingly, the baseline becomes less effective as an NPU scales, similar to the results of PE utilizations. DM-NPU and Planaria achieve similar utilizations as both distribute the DRAM bandwidth and GB capacity to DNNs in a fair manner.

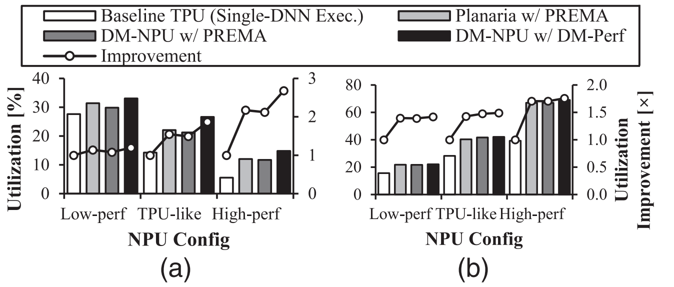

Fig. 13. Improvements in NPU hardware resource utilizations over the baseline TPU with the spatial-multitasking workloads. (a) PEs. (b) Off-chip DRAM bandwidth.

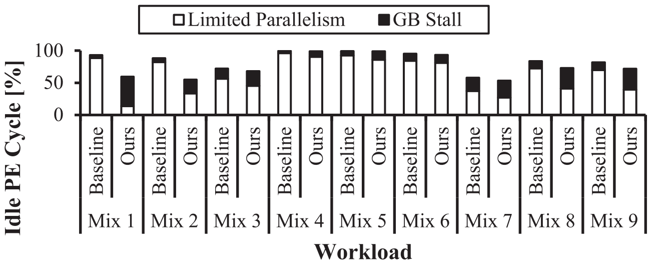

Fig. 14. Idle PE cycle breakdowns of the baseline TPU and DM-NPU with the spatial-multitasking workloads using the TPU-like configuration.

Although DM-NPU significantly improves the PE and DRAM bandwidth utilization, the PEs are not relatively utilized compared to the memory bandwidth. To investigate the cause of PE under-utilization, we break down the PE idle cycles of the baseline TPU and DM-NPU, according to the causes of under-utilization (Fig. 14). Two cases incur PE under-utilization - when a tile cannot cover whole allocated PEs (*Limited Parallelism*) and when the input data is not ready (*GB Stall*). We can verify that DM-NPU effectively resolves the under-utilization due to limited parallelism the baseline suffers. For workload Mix 4-6, the improvement is very small but it is not the DM-NPU's problem; the DNN composing these workloads has too low parallelism, showing at most 7.70% PE utilization. The major cause of PE under-utilization of DM-NPU is GB stall, as multiple DNNs access off-chip DRAM concurrently. Still, DM-NPU outperforms the baseline TPU because reducing limited parallelism outweighs the increase in GB stalls.

TABLE IV
EVALUATED SINGLE-LAYER MICROBENCHMARKS

#	DNN, #Layer	ih × iw × ic	fh × fw × oc	Stride
MB 1	AlexNet, #1	227×227×3	11×11×64	4
MB 2	AlexNet, #2	27×27×64	5×5×192	1
MB 3	AlexNet, #3	13×13×384	3×3×256	1
MB 4	AlexNet, #4	13×13×256	3×3×256	1
MB 5	ResNet-50, #12	56×56×256	1×1×128	2
MB 6	ResNet-50, #13	28×28×128	3×3×128	1
MB 7	ResNet-50, #14	28×28×128	1×1×512	1
MB 8	ResNet-50, #15	56×56×256	1×1×512	2

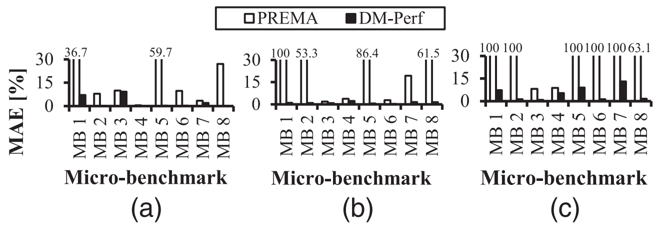

Fig. 15. Microbenchmark modeling accuracies of PREMA and DM-Perf. (a) Low-performance. (b) TPU-like. (c) High-performance.

E. Accurate NPU Performance Modeling

Aimed at evaluating the performance modeling accuracy of DM-Perf, we compare DM-Perf's modeling accuracy against that of PREMA [17], the state-of-the-art systolic-array NPU performance model. As the accuracy metric, we employ the Mean Absolute Error (MAE) which quantifies the difference between the actual and estimated execution times of a DNN. The MAE of a performance model is defined as:

$$MAE = \frac{1}{n}\sum_{i=0}^{n-1} |latency_i - predicted_i| \tag{4}$$

where n denotes the number of layers of a DNN, and $latency_i$ and $predicted_i$ refer to the actual and predicted execution latencies of the i-th layer, respectively.

We first assess a set of microbenchmarks to evaluate the single-layer execution time prediction accuracy of DM-Perf and PREMA. The single-layer microbenchmarks, shown in Table IV are collected from AlexNet and ResNet-50 and exhibit diverse input activation, filter, and stride sizes. Fig. 15 shows the MAEs of DM-Perf and PREMA with the microbenchmarks and the three NPU architecture configurations. DM-Perf achieves higher single-layer execution time modeling accuracy than PREMA; DM-Perf and PREMA achieves MAEs of 2.91% and 44.36%, respectively.

Fig. 16(a) shows the MAEs of DM-Perf and PREMA when estimating the single-DNN execution times with the three NPU architecture configurations. DM-Perf achieves MAEs of 2.3%, 1.4%, and 3.2% with the low performance (Low-Perf), TPU-like, and high performance (High-Perf) configurations, respectively. PREMA, however, suffers from higher MAEs of 25.2%, 41.1%, and 56.3% with Low-Perf, TPU-like, and High-Perf, respectively. PREMA incurs even higher MAEs when the amount of NPU hardware resources increases as it lacks accurate tile and data-reuse modeling.

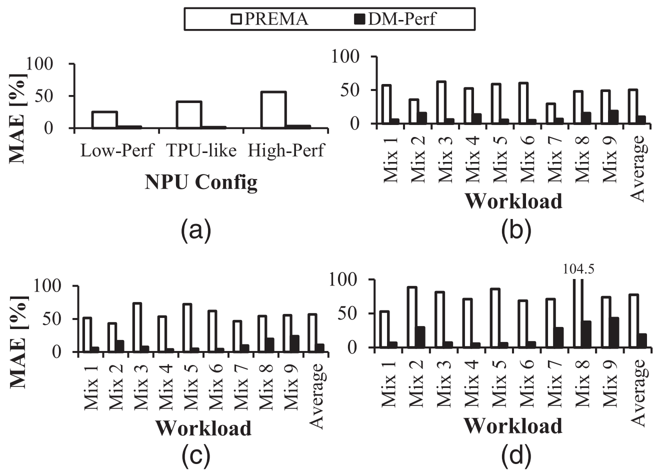

Fig. 16. Modeling accuracies of PREMA and DM-Perf with the spatial-multitasking workloads. (a) Single DNN. (b) Low-performance. (c) TPU-like. (d) High-performance.

Fig. 16(b–d) shows the MAEs of DM-Perf and PREMA with the spatial-multitasking workloads and the three NPU architecture configurations. DM-Perf achieves MAEs of 10.2%, 10.7%, and 18.9% with Low-Perf, TPU-like, and High-Perf configurations, respectively. PREMA, on the other hand, achieves MAEs of 50.3%, 56.8%, and 77.5% with Low-Perf, TPU-like, and High-Perf configurations, respectively, due to its lack of contention modeling. Both DM-Perf and PREMA incur higher MAEs than single-DNN executions due to the contentions caused by spatial multitasking; the actual memory fetch latency increases due to the shared off-chip DRAM bandwidth, and the reduced per-DNN GB capacity provides fewer data reuse opportunities. The MAEs of the two models also become higher with larger amounts of NPU hardware resources. As the PE size increases, the amount of data required to fetch from the off-chip DRAM for executing a tile increases, and thus the DRAM bandwidth contention becomes more significant. The MAEs with Mixes 8 and 9 are higher than those of the other workloads. The two Mixes co-locate four DNNs, incurring more contention on the shared off-chip DRAM bandwidth. Still, for all the configurations and workloads, DM-Perf achieves significantly lower MAEs than PREMA.

F. Energy Consumption

Our energy model reports that the energy consumption of DM-NPU with DM-Perf is slightly higher than that of Planaria with PREMA. DM-NPU with DM-Perf consumes 6.5% more energy in geometric mean over Planaria with PREMA. The increase in energy consumption is due to the improvement in performance; DM-NPU performs more MAC operations per unit time than Planaria, but our experimental setup simulates the same number of cycles (i.e., one billion cycles) for both Planaria and DM-NPU. We claim that the improvements in STP, ANTT, and NPU hardware resource utilization of DM-NPU with DM-Perf are sufficiently large enough to offset the increase in the energy consumption.

TABLE V
PRE-PnR CHIP AREA FOOTPRINTS OF LOW-PERFORMANCE TPU AND DM-NPU IMPLEMENTATIONS SYNTHESIZED WITH THE 7-NM ASAP7 PDK

Component	TPU	DM-NPU	Overhead
PE	0.1697 mm^2	0.2024 mm^2	+0.0327 mm^2 (+19.27%)
SDS+WB	0.0543 mm^2	0.0606 mm^2	+0.0063 mm^2 (+11.66%)
ACCQ	0.0320 mm^2	0.0357 mm^2	+0.0037 mm^2 (+11.44%)
Others	0.3294 mm^2	0.3294 mm^2	-
Total	0.5854 mm^2	0.6280 mm^2	+0.0427 mm^2 (+7.29%)

G. Hardware Implementation Costs

To evaluate the hardware implementation costs of DM-NPU, we implement the baseline TPU and DM-NPU employing the low-performance NPU architecture configuration at the register-transfer level (RTL) using Verilog, and compare their area and power consumptions with a 7-nm cell library. The RTL implementations faithfully model the PEs, ACCQs, SDS, and WB, which are the four major NPU hardware components DM-NPU extends. We synthesize the RTL implementations with ASAP7 [25], a 7-nm predictive Process Design Kit (PDK), and perform Place-and-Route (PnR) using the OpenROAD Flow [26], [27], an open-source RTL-to-GDSII tool. When synthesizing and performing PnR on the RTL implementation, we set 1 GHz as the target clock frequency. For the GB-related costs, we model a 22-nm GB using CACTI 7 [28] and scale the costs down to 7-nm.

The synthesis results show that implementing DM-NPU on top of the TPU incurs a chip area overhead of 7.29%. Table V shows the chip areas of the TPU and DM-NPU after synthesizing their RTL implementations with ASAP7. Most of the overhead comes from the extended PEs as DM-NPU extends each of the PEs to include a lifetime counter and four multiplexers. SDS and WB require additional queues and pipelined wires for feeding the input activations and filter weights to the right- and bottom-most PEs, respectively.

Fig. 17 shows the layouts of the TPU and DM-NPU after performing PnR with the OpenROAD Flow and ASAP7. The post-PnR results show that DM-NPU, compared to the TPU, requires 4.83% larger chip area and incurs 8.10% higher power consumption. Note that the post-PnR chip area overhead is relatively smaller than that of the synthesis (i.e., pre-PnR). We believe that the difference is due to the hierarchical synthesis feature of the OpenROAD Flow. We enabled the hierarchical synthesis feature for the pre-PnR results to report the per-component overheads, but disabled the feature when generating the post-PnR results.

Despite the increases in the chip area and power consumption, we claim that the increases are acceptable given the large spatial-multitasking performance improvements offered by the combination of DM-NPU and DM-Perf. Yet, one can reduce the hardware implementation costs by employing simplified alternative DM-NPU designs. For example, the GB, WB, and SDS unit can be split into two and get placed on both sides of the systolic array to allow the PEs access the closer GB, WB, and SDS unit. However, such a design would lower the performance of single-DNN executions on DM-NPU as only up to 50% of the GB, WB, and SDS unit can be allocated to a DNN.

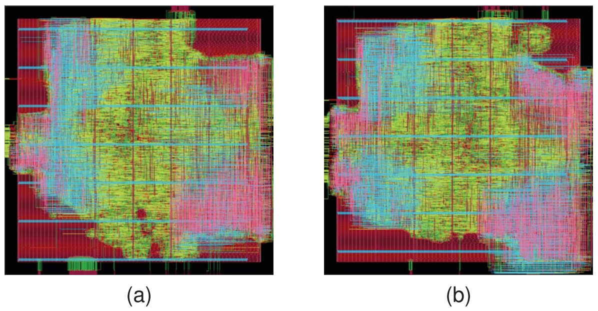

Fig. 17. Post-PnR layouts of low-performance TPU and DM-NPU implementations produced by OpenROAD Flow with the 7-nm ASAP7 PDK. (a) TPU. (b) DM-NPU.

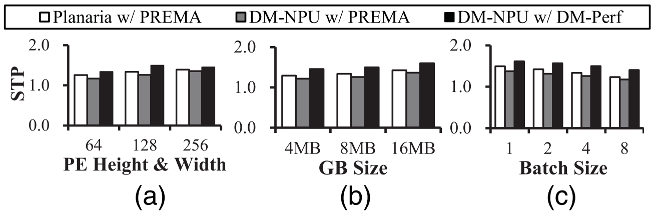

Fig. 18. Sensitivity of the multitasking performance of the three combinations of NPU architecture and performance model to NPU parameters. (a) PE count. (b) GB capacity. (c) Batch size.

VI. SENSITIVITY STUDIES

A. Processing Element Count

We analyze how sensitive the multitasking performances of the three combinations of NPU architecture and performance model are with varying PE counts. Fig. 18(a) shows the geometric-mean STP value achieved by the three combinations with the nine spatial-multitasking workloads when altering the PE count on the TPU-like NPU architecture configuration. For DM-NPU with DM-Perf, decreasing the PE counts results in lower STP as co-located DNNs get allocated fewer PEs. On the other hand, increasing the PE count slightly decreases the multitasking performance. We find that the DNNs suffer from the limited per-DNN GB size as the amount of data for utilizing the increased number of PEs also increases. This results in fewer data reuse opportunities, increasing the DNN execution latency. Still, DM-NPU with DM-Perf outperforms both Planaria with PREMA and DM-NPU with PREMA for all the evaluated PE counts.

B. Global Buffer Size

We now examine how the performance changes with respect to the GB size. The GB size is an important design parameter as it determines the data reuse opportunities of co-located DNNs. Fig. 18(b) shows how the geometric-mean STP values of the three combinations change with varying GB sizes. Reducing the GB size to 4 MB reduces the prefetch and data reuse opportunities, incurring higher DNN execution latency and lower STP. Increasing the GB size to 16 MB improves the STP as the per-DNN GB size increases and the DNNs can exploit more prefetch and data reuse opportunities. However, we find that the performance is not highly sensitive to the GB size as the PE count, which determines the size of the data to be loaded to the GB, remains as-is.

C. Batch Size

We analyze how the performance changes with respect to the batch sizes of the spatial-multitasking workloads. A DNN's batch size determines the number of operations the DNN incurs. Employing large batch sizes increases the amount of computation, decreasing the potential of spatial multitasking due to fewer idle PE cycles. Fig. 18(c) shows the performance of the three combinations with varying batch sizes. Increasing the batch size increases the amount of computation of DNNs; however, as the amount of available NPU hardware resources remains as-is, the DNN execution latency increases and the STP decreases. On the other hand, decreasing the batch sizes results in lower NPU hardware resource utilization, allowing spatial multitasking to achieve higher STP by exploiting the idle resources. Still, for all the batch sizes, DM-NPU with DM-Perf performs the best.

VII. Related Work

A. Spatial Multitasking on Systolic-Array NPUs

Modern systolic-array NPUs are equipped with abundant hardware resources which are difficult to be fully utilized with single-DNN executions. To fully exploit the NPU hardware resources, prior studies have proposed to implement spatial multitasking on the NPUs. Planaria [12] partitions and distributes the NPU hardware resources to co-located DNNs by grouping PEs into sub-arrays and by distributing the sub-arrays to the DNNs. To group the PEs into the sub-arrays, Planaria requires all-to-all high-radix crossbars which incur significant hardware implementation costs. On the other hand, DM-NPU enables fine-grained systolic-array allocations by reversing the dataflows of co-located DNNs. Herald [29] employs multiple sub-accelerators having different PE hierarchies (e.g., reduction tree, systolic array), and then dispatches a DNN to the sub-accelerator expected to achieve the lowest execution latency. Herald achieves spatial multitasking by dispatching different DNNs to different sub-accelerators; however, DM-NPU only extends the systolic array and does not require the sub-accelerators.

B. Temporal Multitasking on Systolic-Array NPUs

An alternative to spatial multitasking is temporal multitasking which makes co-located DNNs share the NPU hardware resources in a time-sharing manner. Temporal multitasking can improve the multitasking performance by overlapping the computation of one DNN with the memory accesses of the other DNNs. Baek et al. [18] propose AI-MT, a temporal multitasking architecture for the simultaneous execution of compute- and memory-intensive DNNs. Layerweaver [30] is a layer-wise time-multiplexing scheduler designed to achieve high STP. While AI-MT requires user-defined thresholds, Layerweaver does not require the effort of extensive parameter tuning. The key idea of the studies is that the DNNs having different characteristics (i.e., compute-intensive and memory-intensive) can be multi-tasked in a temporal manner to improve the NPU performance. DM-NPU, on the other hand, is a spatial-multitasking NPU architecture. Note that DM-NPU can be implemented on top of the temporal-multitasking NPU architectures by extending their systolic arrays.

C. Dynamic PE Reconfiguration

An alternative method to improve the NPU hardware resource utilization is to employ a programmable PE array and dynamically reconfigure the PEs during runtime. Prior studies which utilize the programmable PE array can be classified into two types: 1) reconfiguring the interconnection between multiple sub-arrays of a systolic array to provide the optimized PE height and width for a given DNN [12], and 2) reconfiguring multiple systolic arrays and GBs of equal size for a given DNN multitasking workload [31], [32]. Planaria [12] belongs to the first type as it can adjust its systolic array height and width using the sub-arrays and high-radix all-to-all crossbar. When it comes to spatial multitasking, however, Planaria's resource allocation granularity gets bounded by its sub-array size, allowing Planaria to support coarse-grained spatial multitasking only. An example of the second type is MoCA [31] which implements multiple tiles equipped with a systolic array and a GB of same size, and allocates the tiles to co-located DNNs. Similar to Planaria, MoCA's resource allocation granularity gets bounded by the per-tile systolic array size, making it difficult for MoCA to support fine-grained spatial multitasking. DM-NPU, on the other hand, allows systolic-array allocation boundaries to be set between any adjacent PE rows/columns and enables fine-grained spatial multitasking. Note that DM-NPU can be implemented on top of MoCA by extending the per-tile systolic arrays.

VIII. Conclusion

We proposed DM-NPU, a systolic-array NPU architecture which enables fine-grained spatial multitasking while incurring low hardware implementation costs. DM-NPU reverses the dataflows of DNNs, so that the systolic-array allocation boundaries can be set between any adjacent PE rows and columns. We then proposed DM-Perf, an accurate performance model which uses the per-layer profiles of a DNN to capture the shared NPU hardware resource contentions caused by spatial multitasking. Our experiments using nine multitasking workloads show that DM-NPU with DM-Perf significantly improves the multitasking performance over the state-of-the-art Planaria with PREMA.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. NIPS*, 2012, pp. 1106–1114.

[2] C. Szegedy et al., "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit. (CVPR)*, 2015, pp. 1–9.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit. (CVPR)*, 2016, pp. 770–778.
[4] A. Vaswani et al., "Attention is all you need," in *Proc. NIPS*, 2017, pp. 5998–6008.
[5] X. He et al., "Neural collaborative filtering," in *Proc. WWW*, 2017, pp. 173–182.
[6] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proc. IEEE*, vol. 105, no. 12, pp. 2295–2329, Dec. 2017.
[7] H. T. Kung, "Why systolic architectures?" *Computer*, vol. 15, no. 1, pp. 37–46, Jan. 1982.
[8] N. P. Jouppi et al., "In-datacenter performance analysis of a tensor processing unit," in *Proc. ACM/IEEE 44th Ann. Int. Symp. Comput. Archit. (ISCA)*, 2017, pp. 1–12.
[9] N. P. Jouppi et al., "Ten lessons from three generations shaped Google's TPUv4i," in *Proc. ACM/IEEE 48th Ann. Int. Symp. Comput. Archit. (ISCA)*, 2021, pp. 1–14.
[10] J.-W. Jang et al., "Sparsity-aware and re-configurable NPU architecture for Samsung flagship mobile SoC," in *Proc. ACM/IEEE 48th Ann. Int. Symp. Comput. Archit. (ISCA)*, 2021, pp. 15–28.
[11] B. Liu et al., "Addressing the issue of processing element under-utilization in general-purpose systolic deep learning accelerators," in *Proc. ASP-DAC*, 2019, pp. 733–738.
[12] S. Ghodrati et al., "Planaria: Dynamic architecture fission for spatial multi-tenant acceleration of deep neural networks," in *Proc. 53rd Ann. IEEE/ACM Int. Symp. Microarchit. (MICRO)*, 2020, pp. 681–697.
[13] S. I. Venieris, C.-S. Bouganis, and N. D. Lane, "Multi-DNN accelerators for next-generation AI systems," *Computer*, vol. 56, no. 3, pp. 70–79, Mar. 2022.
[14] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," in *Proc. ACM/IEEE 43rd Ann. Int. Symp. Comput. Archit. (ISCA)*, 2016, pp. 367–379.
[15] J. Ross, "Prefetching weights for use in a neural network processor," US Patent 9 805 304, 2017.
[16] A. Samajdar et al., "A systematic methodology for characterizing scalability of DNN accelerators using SCALE-Sim," in *Proc. ISPASS*, 2020.
[17] Y. Choi and M. Rhu, "PREMA: A predictive multi-task scheduling algorithm for preemptible neural processing units," in *Proc. HPCA*, 2020, pp. 220–233.
[18] E. Baek, D. Kwon, and J. Kim, "A multi-neural network acceleration architecture," in *Proc. ACM/IEEE 47th Ann. Int. Symp. Comput. Archit.* (ISCA), 2020, pp. 940–953.
[19] K. Siu et al., "Memory requirements for convolutional neural network hardware accelerators," in *Proc. IEEE Int. Symp. Workload Characterization (IISWC)*, 2018, pp. 111–121.
[20] S. Li, Z. Yang, D. Reddy, A. Srivastava, and B. Jacob, "DRAMsim3: A cycle-accurate, thermal-capable DRAM simulator," *IEEE Comput. Archit. Lett.*, vol. 19, no. 2, pp. 106–109, Jul.–Dec. 2020.
[21] Y. N. Wu, J. S. Emer, and V. Sze, "Accelergy: An architecture-level energy estimation methodology for accelerator designs," in *Proc. ICCAD*, 2019.
[22] P. Mattson et al., "MLPerf: An industry standard benchmark suite for machine learning performance," *IEEE Micro*, vol. 40, no. 2, pp. 8–16, 2020.
[23] V. J. Reddi et al., "MLPerf inference benchmark," in *Proc. ISCA*, 2020, pp. 446–459.
[24] S. Eyerman and L. Eeckhout, "System-level performance metrics for multiprogram workloads," *IEEE Micro*, vol. 28, no. 3, pp. 42–53, May/Jun. 2008.
[25] L. T. Clark et al., "ASAP7: A 7-nm finFET predictive process design kit," *Microelectronics J.*, vol. 53, 2016, pp. 105–115.
[26] T. Ajayi et al., "OpenROAD: Toward a self-driving, open-source digital layout implementation tool chain," in *Proc. GOMACTech*, 2019, pp. 1105–1110.
[27] T. Ajayi et al., "Toward an open-source digital flow: First learnings from the OpenROAD project," in *Proc. 56th ACM/IEEE Design Automat. Conf. (DAC)*, 2019, no. 76, pp. 1–4.
[28] R. Balasubramonian et al., "CACTI 7: New tools for interconnect exploration in innovative off-chip memories," in *ACM TACO*, vol. 14, no. 2, pp. 1–25, 2017.
[29] H. Kwon, L. Lai, M. Pellauer, T. Krishna, Y.-H. Chen, and V. Chandra, "Heterogeneous dataflow accelerators for multi-DNN workloads," in *Proc. IEEE Int. Symp. High-Performance Comput. Architecture (HPCA)*, 2021, pp. 71–83.
[30] Y. H. Oh et al., "Layerweaver: Maximizing resource utilization of neural processing units via layer-wise scheduling," in *Proc. IEEE Int. Sym. High-Performance Comput. Architecture (HPCA)*, 2021, pp. 584–597.
[31] S. Kim et al., "MoCA: Memory-centric, adaptive execution for multi-tenant deep neural networks," in *Proc. IEEE Int. Symp. High-Performance Comput. Architecture (HPCA)*, 2023, pp. 828–841.
[32] E. Baek, E. Lee, T. Kang, and J. Kim, "STfusion: Fast and flexible multi-NN execution using spatio-temporal block fusion and memory management," *IEEE Trans. Comput.*, vol. 72, no. 4, pp. 1194–1207, Apr. 2022.

Jinwoo Choi (Student Member, IEEE) received the B.S. degree in computer science from Yonsei University, Seoul, South Korea. He is currently working toward the Ph.D. degree in computer science with Yonsei University, Seoul, South Korea. His current research interests include computer architecture and system software optimizations.

Yeonan Ha (Student Member, IEEE) received the B.S. degree in computer science from Yonsei University, Seoul, South Korea. He is currently working toward the Ph.D. degree in computer science with Yonsei University, Seoul, South Korea. His current research interests include computer architecture with an emphasis on FPGA acceleration.

Jounghoo Lee (Student Member, IEEE) received the B.S. degree in electrical and electronic engineering from Yonsei University, Seoul, South Korea. He is currently working toward the Ph.D. degree in computer science with Yonsei University, Seoul, South Korea. His current research interests include computer architecture and performance modeling.

Sangsu Lee (Student Member, IEEE) received the B.S. degree in computer science from Yonsei University, Seoul, South Korea. He is currently working toward the Ph.D. degree in computer science with Yonsei University, Seoul, South Korea. His current research interests include computer architecture and FPGA acceleration.

Jinho Lee (Member, IEEE) received the B.S. degree in electrical engineering, and the M.S. and Ph.D. degrees in electrical engineering and computer science from Seoul National University, Seoul, South Korea. He is currently an Assistant Professor with the Department of Electrical and Computer Engineering, Seoul National University, Seoul, South Korea. His current research interests include architectures and system optimizations for deep learning.

Hanhwi Jang (Member, IEEE) received the B.S. and Ph.D. degrees in computer science and engineering from POSTECH, Pohang, South Korea. He is currently an Assistant Professor with the Department of Electrical and Computer Engineering, Ajou University, Suwon, South Korea. His current research interests include computer architecture and performance modeling.

Youngsok Kim (Member, IEEE) received the B.S. and Ph.D. degrees in computer science and engineering from POSTECH, Pohang, South Korea. He is currently an Assistant Professor with the Department of Computer Science, Yonsei University, Seoul, South Korea. His research interests include computer architecture and system software, with an emphasis on processor microarchitecture, performance modeling, and architecture-conscious system optimizations.