To appear in the *Proceedings of the 58th ACM/IEEE Design Automation Conference (DAC), 2021*

Dataflow Mirroring: Architectural Support for Highly Efficient Fine-Grained Spatial Multitasking on Systolic-Array NPUs

Jounghoo Lee*[1], Jinwoo Choi*[1], Jaeyeon Kim†, Jinho Lee*†, and Youngsok Kim*†
*Department of Computer Science, Yonsei University
†Department of Artificial Intelligence, Yonsei University
{jounghoolee, jinwoo1029, jaeyeon_kim, leejinho, youngsok}@yonsei.ac.kr

***Abstract*—We present *dataflow mirroring*, architectural support for low-overhead fine-grained systolic array allocation which overcomes the limitations of prior coarse-grained spatial-multitasking Neural Processing Unit (NPU) architectures. The key idea of dataflow mirroring is to reverse the dataflows of co-located Neural Networks (NNs) in horizontal and/or vertical directions, allowing allocation boundaries to be set between any adjacent rows and columns of a systolic array and supporting up to four-way spatial multitasking. Our detailed experiments using MLPerf NNs and a dataflow-mirroring-augmented NPU prototype which extends Google's TPU with dataflow mirroring shows that dataflow mirroring can significantly improve the multitasking performance by up to 46.4%.**

I. Introduction

As the computation load of Neural Networks (NNs) continues to increase, Neural Processing Units (NPUs), the specialized hardware accelerators for NNs, are being actively developed. To achieve highly efficient NN acceleration, a number of NPUs employ a two-dimensional array of homogeneous Processing Elements (PEs), known as a systolic array. Systolic arrays achieve highly efficient matrix multiplication, a key operation of NNs, by making each PE independently compute a partial result using the data from its upstream neighbors and pass the data and/or the partial result downstream. In this way, systolic arrays can exploit the abundant parallelism in matrix multiplication and minimize inter-PE communication cost, making them an attractive choice for NPUs [2], [6].

Aimed at fast single-NN executions, NPUs typically allocate all their hardware resources to only a single NN. This allows NPUs to minimize single-NN execution latency; however, NPUs often achieve low hardware utilization and performance due to the insufficient computation load of lightweight NNs and mismatches between the NNs' computation and systolic arrays [8], [12], [13]. A promising solution for improving NPU hardware utilization and performance is *spatial multitasking* which allocates an NPU's hardware resources to multiple co-located NNs [5]. First, spatial multitasking can improve the hardware utilization by allocating the idle hardware resources of single-NN executions to the other co-located NNs. Second, spatial multitasking also improves the performance (e.g., system throughput, turnaround time) as it allows the NNs to run in parallel using their allocated hardware resources. Due to these significant advantages, spatial multitasking is a highly desirable feature for NPUs.

To maximize the benefits of spatial multitasking, NPUs should support *fine-grained allocation* of their hardware resources to co-located NNs. Unfortunately, we observe that prior NPUs achieve sub-optimal hardware utilization and performance as they lack support for fine-grained systolic array allocation. Planaria [5], the state-of-the-art spatial-multitasking NPU, partitions a systolic array into sub-arrays having the same height and width (e.g., a 128×128 systolic array into four 64×64 sub-arrays) and allocates the sub-arrays to co-located NNs. This makes Planaria support only coarse-grained systolic array allocation as the NNs are not allowed to share the systolic array beyond the sub-array boundary. For Planaria to support fine-grained systolic array allocation, it needs to partition a systolic array into 1×1 sub-arrays; however, such small sub-arrays would incur large design cost (e.g., all-to-all high-radix crossbars). To achieve highly efficient spatial multitasking, we need a new systolic array architecture which supports fine-grained allocation with small design cost.

In this paper, we present *dataflow mirroring*, architectural support for fine-grained and low-overhead allocation of systolic arrays to co-located NNs. Dataflow mirroring enables highly efficient fine-grained allocation of a systolic array at the granularity of single row and column as follows. First, dataflow mirroring prevents any interference between the NNs by making their data flow in reverse directions from an allocation boundary to the borders of the systolic array. Second, to achieve the small allocation granularity with small design cost, dataflow mirroring employs an omni-directional inter-PE network which allows every row and column to be an allocation boundary. Third, dataflow mirroring 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. To summarize, dataflow mirroring satisfies all the design goals of spatial-multitasking NPUs by achieving highly efficient fine-grained systolic array allocation.

We then design a dataflow-mirroring-augmented NPU prototype and a software runtime to evaluate the effectiveness of dataflow mirroring. The prototype achieves highly efficient fine-grained spatial multitasking by extending Google's TPU with fine-grained allocation of the systolic array, on-chip SRAM buffers, and off-chip DRAM bandwidth to co-located NNs. The prototype also supports dynamic hardware resource reallocation upon NN entries and exits by exploiting lightweight preemption support [3]. The software runtime automatically finds the optimal systolic array allocation for the co-located NNs using an accurate NPU performance model [3]. By exploiting the fine-grained resource allocation and preemption capabilities of the prototype, the software runtime dynamically reallocates the hardware resources to maximize the hardware utilization and performance.

Our experiments using MLPerf NNs [9], [10] and detailed cycle-level simulators [7], [11] show that dataflow mirroring can greatly improve the NPU hardware utilization and performance. On a TPU-like hardware configuration, dataflow mirroring improves the multitasking performance by up to 46.4% over the state-of-the-art coarse-grained spatial-multitasking NPU architecture [5]. The improvements come from dataflow mirroring's ability to easily adapt to diverse loads of co-located NNs by allocating the systolic array in a fine-grained manner, whereas the state-of-the-art achieves sub-optimal performance due to its coarse-grained allocation granularity.

In summary, this paper makes the following contributions:

- We propose *dataflow mirroring*, lightweight architectural support

[1]These authors contributed equally to this work.

for fine-grained systolic array allocation. Its key idea is to reverse the dataflows of NNs in horizontal and/or vertical direction.

- We present *FGSpMt-NPU*, a fine-grained spatial-multitasking NPU architecture which extends Google's TPU [6] with dataflow mirroring. Along with fine-grained hardware resource allocation, FGSpMt-NPU supports dynamic re-allocation of the hardware resources upon NN entries and exits with preemption support [3].
- We design a software runtime for FGSpMt-NPU which maximizes the multitasking performance by exploiting the fine-grained systolic array allocation and a lightweight NPU performance model [3].

II. Background & Motivation

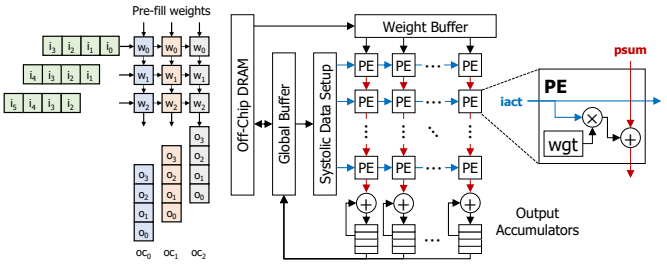
A. Executing Neural Networks on Systolic Arrays

Given an input, a Neural Network (NN) makes a prediction by executing a series of *layers* which perform different operations on the input data. Each layer takes as input four-dimensional input activations (iacts) whose batch size, height, width, and channel count are n, ih, iw, ic, respectively, and produces four-dimensional output activations (oacts) whose size is $n{\times}oh{\times}ow{\times}oc$ where oh, ow, and oc are height, width, and channel count, respectively. Among the layers, convolutional layers have been a primary acceleration target of NPUs as they tend to incur the largest amounts of computation [15]. A convolutional layer slides a filter over the iacts, multiplies the weights of the filter with the corresponding iacts, and accumulates the multiplication results to produce oacts. The convolutional layer can be expressed as $oact[n][oh][ow][oc] = actFun(\sum_{i=0}^{fh-1}\sum_{j=0}^{fw-1}\sum_{c=0}^{ic-1} iact[n][oh+i][ow+j][c] \times weight[i][j][c][oc])$ where fh and fw are the filter height and width, respectively, and $actFun$ is an activation function (e.g., the rectified linear unit). By flattening the iacts using the image-to-column transformation with respect to the filter and oact sizes, the multiplication of the iacts and weights becomes the matrix multiplication of the $(n{\times}oh{\times}ow)$-by-$(fh{\times}fw{\times}ic)$ flattened iact matrix and $(fh{\times}fw{\times}ic)$-by-oc weight matrix. For this reason, matrix multiplication is a key operation of convolutional layers and NPUs are typically designed to accelerate matrix multiplication [15].

For fast NN executions, NPUs typically employ a systolic array of Processing Elements (PEs) to accelerate matrix multiplication. How the systolic array performs matrix multiplication depends on its *dataflow* which defines how the layers' data flow on the systolic array. Fig. 1a shows how the PEs perform matrix multiplication using the Weight-Stationary (WS) dataflow [15]. First, the PEs get pre-filled with their weights. The weights of one filter get distributed to a single PE column, making different PE columns process different output channels. Then, the iacts of the layer get streamed horizontally across the PEs. At each cycle, each PE multiplies its weight with the iact from the PE on the left, accumulates the multiplication result with the partial oact (psum) received from the PE above, and streams the updated psum to the PE below. Once a psum reaches the bottom of the systolic array, the psum gets accumulated to the corresponding output accumulator located below the systolic array. Fig. 1b shows a representative systolic-array NPU architecture implementing the WS dataflow. It consists of a systolic array, the weight buffer for feeding the weights to the systolic array, the global buffer for storing iacts and oacts, the systolic data setup unit for flattening the iacts, and the off-chip DRAM. Each PE has a local buffer to store its weight.

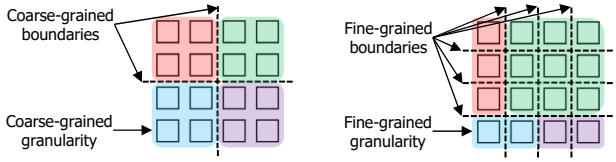
B. Spatial Multitasking on Systolic-Array NPUs

Most NPUs today allocate all their hardware resources to only a single NN at a time to achieve fast single-NN executions. Using the abundant hardware resources, NPUs execute an NN's layers in a sequential manner; however, allocating all the hardware resources to a single NN makes NPUs suffer from low hardware utilization and performance. For example, systolic arrays using the WS dataflow (Fig. 1a) fill their PEs with a $(fh{\times}fw{\times}ic)$-by-oc weight matrix to perform matrix multiplication. In case the weight matrix is smaller than the systolic array on any dimension or the matrix size is not an exact multiple of the systolic array size, a significant portion of the systolic array will remain idle, resulting in a significant performance loss [8], [16]. Moreover, such under-utilization becomes more severe as the systolic array size gets larger [11].



(a) Working model (b) NPU architecture

Fig. 1: Working model and architecture of weight-stationary NPUs



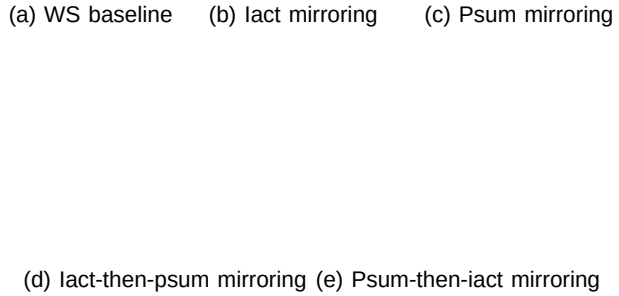
(a) Coarse-grained [5] (b) Fine-grained (Ours)

Fig. 2: Coarse- and fine-grained systolic array allocation

To overcome the low hardware utilization and performance, spatial multitasking which allocates the hardware resources to multiple NNs and concurrently executes the NNs is a highly desirable feature. Recent work [5] proposes to implement spatial multitasking by partitioning an NPU's systolic array into sub-arrays having the same height and width and by allocating the sub-arrays to co-located NNs. Allocating the sub-arrays to the NNs can improve the hardware utilization and performance; however, it lacks support for *fine-grained* systolic array allocation and achieves suboptimal hardware utilization and performance improvements. Fig. 2a illustrates the recent work's sub-array-based allocation of a 4×4 systolic array using 2×2 sub-arrays. First, as the sub-array becomes the allocation granularity, the systolic array can only be partitioned at the coarse-grained allocation boundaries (i.e., the dotted lines Fig. 2a). More fine-grained granularity such as single rows and columns cannot be used as they go beyond the sub-array boundaries. Second, achieving more fine-grained systolic array allocation using the sub-arrays incurs significant design costs as smaller sub-arrays (e.g., 1×1 sub-arrays) require a complex interconnection network between the sub-arrays. The recent work employs all-to-all high-radix crossbars to interconnect the sub-arrays; however, the design cost of the crossbars becomes impractical for NPUs as the number of the sub-arrays increases. Therefore, to achieve highly efficient fine-grained spatial multitasking on NPUs, we need a new low-overhead systolic array architecture supporting fine-grained allocation beyond the sub-array boundaries.

C. Design Goals

Motivated by the low hardware utilization and performance due to the coarse-grained systolic array allocation, we aim to design a new systolic array architecture supporting *fine-grained systolic array allocation* as shown in Fig. 2b. The new systolic array architecture should



(a) WS baseline (b) Iact mirroring (c) Psum mirroring

(d) Iact-then-psum mirroring (e) Psum-then-iact mirroring

Fig. 3: Fine-grained systolic array allocation with data ow mirroring. The arrows indicate the ow of iacts (blue) and psums (red).

satisfy the following design goals. First, the allocation granularity of the systolic array should not be bound to the sub-arrays having the same height and width. Second, the systolic array should support a high number of co-located NNs to maximize the performance of spatial multitasking. Third, its design cost should be small so that the existing NPUs can easily employ the new systolic array architecture.

III. Fine-Grained Systolic Array Allocation

A. Key Idea: Reverse the Data ows of Co-located NNs

We present data ow mirroring, lightweight architectural support for ne-grained systolic array allocation. The key idea of data ow mirroring is to reverse the data ows of co-located NNs in horizontal and/or vertical directions. Reversing the data ows enables ne-grained systolic array allocation and allows a two-dimensional systolic array to co-locate up to four NNs. First, reversing the data ows allows every row and column of a systolic array to be an allocation boundary, enabling ne-grained systolic array allocation. Second, as the data ows can be reversed twice, each in the horizontal and vertical directions, up to three allocation boundaries can be set and up to four NNs to be co-located on the same systolic array. In this way, data ow mirroring achieves all the design goals (Sec. II-C).

Data ow mirroring supports two modes: iact mirroring and psum mirroring, to reverse the data ows. The iact mirroring horizontally reverses the direction of one NN's iacts, whereas the psum mirroring vertically reverses the direction of one NN's psums with respect to a selected allocation boundary. Fig. 3 illustrates the two modes on an example 4 4 systolic array using the WS data ow. The baseline WS data ow streams the iacts in the horizontal direction from left to right and the psums in the vertical direction from top to bottom (Fig. 3a). When two NNs are co-located, the systolic array distributes its PE columns or rows to the two NNs depending on the selected mirroring mode. To distribute the PE columns to the two NNs, the systolic array utilizes the iact mirroring (Fig. 3b). After selecting a vertical allocation boundary, the iacts of one NN ow from left to right, whereas the iacts of the other NN ow right-to-left. Once the iacts reach the allocation boundary, the systolic array discards the iacts so no interference occurs between the two NNs. Meanwhile, the psums of the two NNs ow in the same vertical direction (i.e., from top to bottom) as the psum ows are not reversed. The psum mirroring, on the other hand, reverses the psum ow of one NN (Fig. 3c). With respect to a selected horizontal allocation boundary, the psums of one NN ows from top to bottom and the psums of the other NN ow toward the top. The iacts of the two NNs ow left-to-right as usual. In this way, the systolic array can allocate its PE rows to the two NNs.

To support a higher number of co-located NNs, data ow mirroring can apply the iact mirroring and psum mirroring in a hierarchical manner. By reversing the data ows in both the horizontal and vertical directions using the iact mirroring and psum mirroring, respectively, data ow mirroring can allocate a systolic array to up to four NNs. Fig. 3d and Fig. 3e illustrate how the two mirroring modes can be applied at the same time. As the rst step, a systolic array speci es a global allocation boundary using either the iact mirroring or the psum mirroring. When the global allocation boundary is speci ed with the iact mirroring, the systolic array transforms into two sub-arrays having different numbers of PE columns but the same number of PE rows. The two sub-arrays then can specify their local allocation boundaries using the psum mirroring. Similarly, when the psum mirroring is used to specify the global allocation boundary, the systolic array turns into two sub-arrays having different numbers of PE rows but the same number of PE columns. The two sub-arrays can further be partitioned by setting their local allocation boundaries with the iact mirroring. Note that the same mirroring mode cannot be applied more than once as it would incur interference between the data ows. By allocating each partition of the systolic array to co-located NNs, data ow mirroring allows up to four NNs to be co-located on the same systolic array. Data ow mirroring can support three-way allocation by not specifying a local allocation boundary for one of the two sub-arrays.

B. Advantages of Data ow Mirroring

Data ow mirroring achieves ne-grained systolic array allocation by reversing the data ows of co-located NNs using the iact mirroring and psum mirroring. The ne-grained systolic array allocation not only achieves all the design goals, but also provides several advantages over the prior coarse-grained systolic array allocation. First, reversing the data ows enables allocation boundaries to be set between any adjacent PE rows and columns, making the boundaries not bound to xed height and width. Second, up to four NNs can be co-located on the same systolic array by using both the iact mirroring and psum mirroring. Third, implementing data ow mirroring on the existing systolic-array NPUs requires only a modest amount of architectural modi cation, allowing the NPUs to easily employ data ow mirroring. In summary, data ow mirroring can serve as a key component for achieving highly ef cient spatial multitasking on NPUs due to its ne-grained allocation capability, high number of co-located NNs, and low implementation cost.

IV. Highly Efficient Fine-Grained Spatial Multitasking on Systolic-Array NPUs Using Dataflow Mirroring

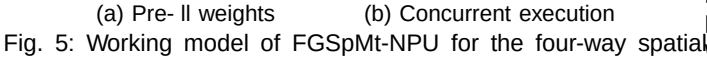
We now design and present FGSpMt-NPU, a Fine-Grained Spatial-Multitasking NPU architecture which implements data ow mirroring. We show that data ow mirroring is easy to implement on the existing systolic-array NPUs by proposing lightweight architectural modi cations which augment Google's TPU [6] with data ow mirroring.

A. Architectural Extension for Data ow Mirroring

Implementing data ow mirroring on systolic-array NPUs introduces key implementation challenges as follows. First, to reverse the iact and psum ows of co-located NNs, the PEs of a systolic array should be able to forward their iacts left and right, and their psums to the upper and lower PEs. Second, the PEs should not horizontally forward the iacts of the NNs beyond the allocated systolic array

(a) Overall architecture (b) PE architecture

Fig. 4: FGSpMt-NPU architecture implementing data ow mirroring



(a) Pre- ll weights (b) Concurrent execution

Fig. 5: Working model of FGSpMt-NPU for the four-way spatial multitasking using the psum-then-iact mirroring shown in Fig. 3e

regions to prevent any interference between the NNs. Third, for ne-grained systolic array allocation, placing an allocation boundary between any adjacent PE rows and columns should be supported.

To solve the challenges, FGSpMt-NPU employs an omni-directional inter-PE network and associates a lifetime counter to each horizontal datum (e.g., iacts for the WS data ow). Fig. 4 shows the microarchitecture of FGSpMt-NPU which extends Google's TPU [6] employing the WS data ow to support data ow mirroring. First, the omni-directional inter-PE network enables bi-directional horizontal and vertical communication between the PEs. For the WS data ow, a PE can forward its iacts to both the left and right PEs and its psums to both the upper and lower PEs. By enabling bi-directional inter-PE communication, the omni-directional inter-PE network resolves the rst implementation challenge. Second, the lifetime counter associated to a horizontal datum speci es how far the datum can ow horizontally on the systolic array. On FGSpMt-NPU, a lifetime counter speci es the number of PE columns the associated iact should ow. After performing its multiply-accumulate (MAC) operation with a received iact, each PE rst decrements the lifetime counter of the iact by one. Then, the PE examines the value of the updated lifetime counter. If the lifetime counter is zero, the PE discards the iact as it is no longer necessary. Otherwise, the PE forwards the iact with the updated lifetime counter to the target adjacent PE. By controlling the validity of each iact using the associated lifetime counter, FGSpMt-NPU resolves the second and third key implementation challenges.

In addition, FGSpMt-NPU extends other hardware components to feed iacts from both the left and right borders of a systolic array and to collect psums from both the top and bottom of the systolic array. First, to concurrently pre- ll the weights of co-located NNs, FGSpMt-NPU adds extra wires from the Weight Buffer (WB) unit to the PEs through not only the top-most PE row, but also the bottom-most PE row. Second, FGSpMt-NPU extends the Systolic Data Setup (SDS) unit of the baseline TPU architecture so that the iacts of the NNs can be fed to the systolic array through both the left-most and the right-most PE columns. Third, FGSpMt-NPU allows the psums which have reached not only the bottom-most PE row, but also the top-most PE row to be accumulated to their target output accumulators.

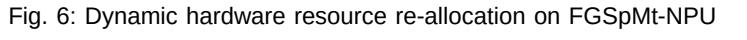


Fig. 6: Dynamic hardware resource re-allocation on FGSpMt-NPU

These architectural extensions allow FGSpMt-NPU to faithfully implement data ow mirroring on systolic-array NPUs. Fig. 5 demonstrates how FGSpMt-NPU supports the iact mirroring and psum mirroring at the same time by using Fig. 3e as an example scenario. When pre- lling the weights of four co-located NNs, FGSpMt-NPU exploits the extended wiring from the WB unit to the systolic array to concurrently pre- ll the PEs with their weights (Fig. 5a). Then, FGSpMt-NPU exploits the omni-directional inter-PE network and the extended SDS unit to execute the co-located NNs in parallel (Fig. 5b). By referring to the lifetime counters associated with the iacts, the iacts of the co-located NNs do not interfere with each other as the iacts get discarded by the PEs at the allocation boundaries. After that, the psums which reach the top- and bottom-most PE rows concurrently populate the corresponding output accumulators using the extended wires from the systolic array to the output accumulators.

B. Fine-Grained On-Chip SRAM & Off-Chip DRAM Allocation

Performing spatial multitasking on systolic-array NPUs demands allocation of not only a systolic array, but also the WB, Global Buffer (GB), and off-chip DRAM bandwidth. The WB and GB store the weights and iacts for executing an NN's layer, respectively, and the off-chip DRAM bandwidth determines how fast an NN can retrieve the necessary weights and iacts from the DRAM to the WB and GB. For simplicity, FGSpMt-NPU allocates the same amount of the WB, GB, and off-chip DRAM bandwidth to co-located NNs. We made this design choice as the focus of FGSpMt-NPU is on the ne-grained allocation of systolic arrays rather than the other hardware resources. FGSpMt-NPU allocates an equal amount of the WB and GB to co-located NNs by distributing the SRAM banks of the WB and GB, and an equal amount of off-chip DRAM bandwidth by making the off-chip DRAM controller fetch the pending memory requests of the NNs in a round-robin manner. After fetching the pending requests, the DRAM controller utilizes its scheduler (e.g., FR-FCFS) to serve the requests.

C. Dynamic Hardware Resource Re-Allocation

One important feature FGSpMt-NPU should implement is dynamic hardware resource re-allocation as co-located NNs can dynamically change due to new NN executions and the completion of the existing NNs. To implement the dynamic re-allocation, FGSpMt-NPU employs a lightweight preemption mechanism for systolic-array NPUs from a recent study [3]. In particular, FGSpMt-NPU implements a variant of DRAIN mechanism of the recent study which waits until the currently-executing layers of the co-located NNs complete their execution. By preempting the currently-executing layers upon an NN entry or exit, FGSpMt-NPU can dynamically re-allocate its hardware resources to a new set of co-located NNs in a timely manner.

Fig. 6 illustrates how FGSpMt-NPU performs dynamic re-allocation when a co-located NN completes its execution. In this example, FGSpMt-NPU is executing four co-located NNs using the psum-then-iact mirroring. Then, as the NN which occupies the bottom-right portion of the systolic array completes its execution, FGSpMt-NPU issues a preemption command to the other co-located

Fig. 7: FGSpMt-NPU software runtime

NNs to reclaim all the hardware resources. After that, FGSpMt-NPU re-allocates its hardware resources to the remaining three NNs and resumes their execution using the iact-then-psum mirroring. Likewise, when a new NN gets co-located on FGSpMt-NPU, FGSpMt-NPU preempts the currently-executing NNs, re-allocates its hardware resources to the new set of NNs, and starts/resumes the NNs' execution.

D. Identifying the Optimal Resource Allocation

Given a set of co-located NNs, FGSpMt-NPU should identify the optimal hardware resource allocation to maximize its target performance. For the purpose, FGSpMt-NPU employs a software runtime which derives the optimal allocation using an accurate systolic-array NPU performance model from a recent study [3]. Fig. 7 illustrates the components and working model of the FGSpMt-NPU software runtime. The FGSpMt-NPU software runtime consists of a topology analyzer and a resource allocator. First, the topology analyzer generates the topology snippets of the NNs (e.g., layer count, per-layer input and output shapes). Then, the resource allocator generates a set of allocation candidates by applying data flow mirroring (i.e., iact/psum/iact-then-psum/psum-then-iact mirroring). For each of the allocation candidates, the resource allocator predicts the per-NN execution latency using the latency prediction model and the topology sniffets. After that, the resource allocator identifies the optimal allocation which maximizes the target multitasking performance such as the highest system throughput (STP) or the lowest average normalized turnaround time (ANTT) [4]. The identified allocation is then sent to FGSpMt-NPU, and FGSpMt-NPU uses the identified allocation to concurrently execute the NNs. By exploiting the lightweight latency prediction model, the resource allocator can quickly identify the optimal allocation, allowing FGSpMt-NPU to easily adapt to dynamic changes in co-located NNs in a timely manner.

V. Evaluation

A. Experimental Setup

To examine the effectiveness of data flow mirroring, we model FGSpMt-NPU by extending SCALE-sim [11], a detailed cycle-level systolic-array NPU architecture simulator. We employ Google's TPU [6] as the baseline NPU hardware configuration by referring to the verified configuration from a prior study [3]. Table I summarizes our simulation parameters and values. For accurate modeling of off-chip DRAM access timing, we use DRAMsim3 [7] for serving FGSpMt-NPU's DRAM accesses (e.g., load iacts from the off-chip DRAM to the GB). We estimate the energy consumption of FGSpMt-NPU with Accelergy [17] for the systolic array, CACTI [1] for the on-chip SRAM accesses, and DRAMsim3 [7] for the off-chip DRAM accesses. Our simulation framework first collects the performance counters required by the energy models from the timing simulation framework. Then, we feed the collected performance counters to the energy models, calculate the per-component energy consumption, and derive the total energy consumption by adding up the per-component energy consumption. The simulation framework also provides an estimated chip area of FGSpMt-NPU. Our prototyping results using a 22-nm cell library show that implementing FGSpMt-NPU on top of the 128 128 TPU increases the chip area by only 14.80 mm^2

TABLE I: FGSpMt-NPU simulation parameters

Parameter	Values
Clock frequency	1 GHz
Systolic array	64 64, 128 128, 256 256
Output accumulators	2048 rows/column
On-chip SRAM buffer	32 banks, 32 B/cycle, 20 MB
Off-chip DRAM	HBM2, 8 channels, 256 GB/s
Computation order	Filter-major [14]
Memory scheme	Working sets of filter and activations [14]

TABLE II: Evaluated MLPerf NNs

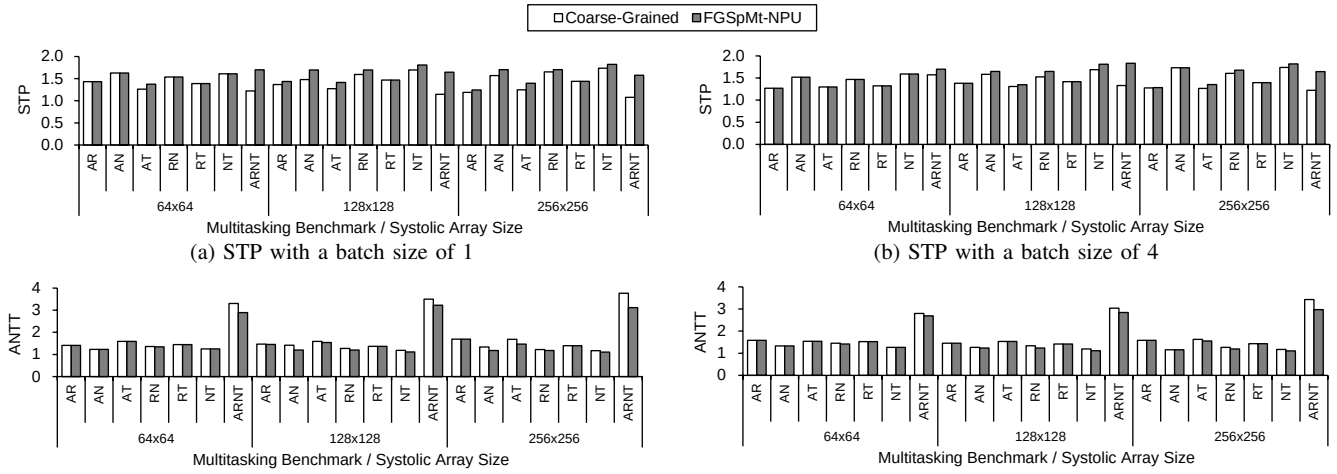
Name	# of Layers	Input Size (HxWxC)	PE Utilization (4 batches)		
			64 64	128 128	256 256
AlexNet	8	227x227x3	20.0%	10.9%	5.7%
ResNet50	54	224x224x3	70.1%	45.9%	25.8%
NCF	8	1x1x138000	0.8%	0.2%	0.1%
Transformer	891	1x1x33708	2.0%	0.9%	0.3%

We model the state-of-the-art coarse-grained spatial-multitasking NPU architecture [5] by partitioning a systolic array into four sub-arrays having the same height and width. Then, for a given set of co-located NNs, our model allocates an equal number of the sub-arrays to the NNs to faithfully model the coarse-grained systolic array allocation. For example, each NN utilizes one of the sub-arrays when there are four co-located NNs as there are four sub-arrays. On the other hand, FGSpMt-NPU identifies and utilizes the optimal fine-grained systolic array allocation using data flow mirroring.

To quantify the NPU performance, we employ two widely-used multitasking performance metrics: STP and ANTT [4]. STP measures the number of NN iterations completed per unit of time and is a system-oriented higher-is-better metric. Higher STP values can be achieved when the NNs' progress does not get affected by multitasking. ANTT, on the other hand, is a lower-is-better user-oriented metric which measures the turnaround-time slowdowns due to multitasking. Achieving the lowest-possible ANTT value (i.e., 1) indicates that the turnaround times are not affected by multitasking. As the multitasking benchmarks, we use four representative MLPerf NNs [9], [10] with batch sizes of 1 and 4. Table II shows the key characteristics of the NNs and their PE utilizations with a batch size of 4. Using the NNs, we generate two- and four-way multitasking benchmarks by co-locating different NNs on the same NPU. In this way, we obtain six two-way benchmarks and one four-way benchmarks. Then, for each benchmark, we simulate 100 million clock cycles of the state-of-the-art and FGSpMt-NPU to ensure that all the NNs of the benchmark complete at least one iteration.

B. High System-Perceived Performance

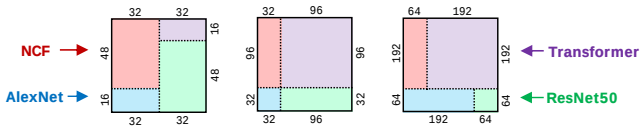
We first examine the STP improvements of FGSpMt-NPU over the state-of-the-art coarse-grained NPU architecture. As FGSpMt-NPU can support all the allocation of the state-of-the-art by implementing data flow mirroring which allows fine-grained systolic-array allocation capability, FGSpMt-NPU should achieve an STP value higher than or equal to that of the state-of-the-art for all the benchmarks. Experimental results show that FGSpMt-NPU can improve STP by up to 46.4% over the state-of-the-art by allowing more fine-grained allocation of a systolic array. Fig. 8a and Fig. 8b show the STP values of FGSpMt-NPU and the state-of-the-art for the multitasking benchmarks with varying systolic array and batch sizes. FGSpMt-NPU tends to achieve larger STP improvements as the systolic array size increases; FGSpMt-NPU improves STP by 46.4% and 34.8% over the state-of-the-art with batch sizes of 1 and 4, respectively, for the four-way multitasking benchmark (i.e., ARNT) on a 256 256 systolic array. Our analysis reveals that, for smaller systolic arrays, allocating an equal number of PEs tends to achieve the highest STP values as the NNs can efficiently utilize the smaller systolic arrays (Table II); however, larger systolic arrays make the NNs difficult to



(a) STP with a batch size of 1 (b) STP with a batch size of 4

(c) ANTT with a batch size of 1 (d) ANTT with a batch size of 4

Fig. 8: STP and ANTT of the state-of-the-art and FGSpMt-NPU with the multitasking benchmarks. Higher STP and lower ANTT are better. A, R, N, and T represent AlexNet, ResNet50, NCF, and Transformer, respectively (e.g., AR: co-location of AlexNet and ResNet50).



(a) 64×64 (b) 128×128 (c) 256×256

Fig. 9: Identified STP-optimal fine-grained allocation using dataflow mirroring for the ARNT benchmark with a batch size of 1

fully utilize the PEs, favoring FGSpMt-NPU over the state-of-the-art. Fig. 9 illustrates the optimal fine-grained systolic array allocation which achieves the highest STP values for different systolic array sizes. To summarize, FGSpMt-NPU greatly improves the system-perceived performance using its fine-grained allocation capability.

Our energy model reports that FGSpMt-NPU can increase the energy consumption by 8.9% in the geometric mean over the state-of-the-art with a 128×128 systolic array. The increase in the energy consumption is due to the increased performance; the more MAC operations FGSpMt-NPU executes per unit time, the more energy it consumes during the same amount of time (i.e., 100 million cycles). We claim that the large STP improvements are sufficiently appealing against the increase in the energy consumption.

C. High User-Perceived Performance

We now examine the improvements in the user-perceived performance by comparing the ANTT values of FGSpMt-NPU and the state-of-the-art (Fig. 8c and Fig. 8d). As ANTT is a lower-is-better metric, the results indicate that FGSpMt-NPU improves the performance by up to 17.2%. By exploiting its fine-grained systolic array allocation capability, FGSpMt-NPU can allocate its systolic array in a way that minimizes ANTT. FGSpMt-NPU reduces ANTT by 17.2% and 13.4% with batch sizes of 1 and 4, respectively, over the state-of-the-art for the four-way multitasking benchmark on a 256×256 systolic array. On the other hand, the state-of-the-art suffers from high ANTT values as its coarse-grained systolic array allocation cannot achieve the optimal fine-grained systolic array allocation.

VI. Conclusion

We proposed dataflow mirroring, lightweight architectural support for fine-grained systolic array allocation. By reversing the dataflows of co-located NNs, dataflow mirroring allows allocation boundaries to be set between any adjacent PE rows and columns. Then, we designed FGSpMt-NPU, a highly efficient spatial-multitasking NPU architecture which implements dataflow mirroring to achieve higher hardware utilization and performance over the existing coarse-grained spatial-multitasking NPU architecture. By enabling fine-grained distribution of the systolic array to co-located NNs, FGSpMt-NPU can greatly improve the multitasking performance over the state-of-the-art.

Acknowledgements

This work was supported by the National Research Foundation of Korea (NRF) grant (No. 2020R1F1A1069742) and Institute of Information & Communications Technology Planning & Evaluation (IITP) grant (No. 2020-0-01361, Artificial Intelligence Graduate School Program(Yonsei University)) funded by the Korea government (MSIT), and the Yonsei University Research Fund (2020-22-0511, 2021-22-0001). Youngsok Kim is the corresponding author of this paper.

References

[1] R. Balasubramonian et al., "CACTI 7: New Tools for Interconnect Exploration in Innovative Off-Chip Memories," *ACM TACO*, 2017.
[2] Y.-H. Chen et al., "Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks," in *ISCA*, 2016.
[3] Y. Choi and M. Rhu, "PREMA: A Predictive Multi-task Scheduling Algorithm For Preemptible NPUs," in *HPCA*, 2020.
[4] S. Eyerman and L. Eeckhout, "System-Level Performance Metrics for Multiprogram Workloads," *IEEE Micro*, 2008.
[5] S. Ghodrati et al., "Planaria: Dynamic Architecture Fission for Spatial Multi-Tenant Acceleration of Deep Neural Networks," in *MICRO*, 2020.
[6] N. P. Jouppi et al., "In-Datacenter Performance Analysis of a Tensor Processing Unit," in *ISCA*, 2017.
[7] S. Li et al., "DRAMsim3: A Cycle-Accurate, Thermal-Capable DRAM Simulator," *IEEE CAL*, 2020.
[8] B. Liu et al., "Addressing the Issue of Processing Element Under-Utilization in General-Purpose Systolic Deep Learning Accelerators," in *ASP-DAC*, 2019.
[9] P. Mattson et al., "MLPerf: An Industry Standard Benchmark Suite for Machine Learning Performance," *IEEE Micro*, 40(2), 2020.
[10] V. J. Reddi et al., "MLPerf Inference Benchmark," in *ISCA*, 2020.
[11] A. Samajdar et al., "A Systematic Methodology for Characterizing Scalability of DNN Accelerators using SCALE-Sim," in *ISPASS*, 2020.
[12] G. Shomron et al., "SMT-SA: Simultaneous Multithreading in Systolic Arrays," *IEEE CAL*, 2019.
[13] G. Shomron and U. Weiser, "Non-Blocking Simultaneous Multithreading: Embracing the Resiliency of Deep Neural Networks," in *MICRO*, 2020.
[14] K. Siu et al., "Memory Requirements for Convolutional Neural Network Hardware Accelerators," in *IISWC*, 2018.
[15] V. Sze et al., "Efficient Processing of Deep Neural Networks: A Tutorial and Survey," *IEEE Micro*, 2017.
[16] X. Wang et al., "SNA: A Siamese Network Accelerator to Exploit the Model-Level Parallelism of Hybrid Network Structure," in *DATE*, 2020.
[17] Y. N. Wu et al., "Accelergy: An Architecture-Level Energy Estimation Methodology for Accelerator Designs," in *ICCAD*, 2019.