SHENG-CHUN KAO, Georgia Institute of Technology, USA HYOUKJUN KWON, Georgia Institute of Technology, USA MICHAEL PELLAUER, NVIDIA, USA ANGSHUMAN PARASHAR, NVIDIA, USA TUSHAR KRISHNA, Georgia Institute of Technology, USA

The high efficiency of domain-specific hardware accelerators for machine learning (ML) has come from *specialization*, with the trade-off of less configurability/ flexibility. There is growing interest in developing *flexible* ML accelerators to make them future-proof to the rapid evolution of Deep Neural Networks (DNNs). However, the notion of accelerator flexibility has always been used in an informal manner, restricting computer architects from conducting systematic apples-to-apples design-space exploration (DSE) across trillions of choices. In this work, we formally define accelerator flexibility and show how it can be integrated for DSE.

Specifically, we capture DNN accelerator flexibility across four axes: tiling, ordering, parallelization, and array shape. We categorize existing accelerators into 16 classes based on their axes of flexibility support, and define a precise quantification of the degree of flexibility of an accelerator across each axis. We leverage these to develop a novel flexibility-aware DSE framework. We demonstrate how this can be used to perform first-of-their-kind evaluations, including an isolation study to identify the individual impact of the flexibility axes. We demonstrate that adding flexibility features to a hypothetical DNN accelerator designed in 2014 improves runtime on future (i.e., present-day) DNNs by 11.8× geomean.

 $\label{eq:CCS} \textit{Concepts:} \bullet \textit{Computer systems organization} \rightarrow \textit{Data flow architectures}; \textit{Neural networks}.$

Additional Key Words and Phrases: Accelerator; Hardware Flexibility; DNN Workloads

ACM Reference Format:

Sheng-Chun Kao, Hyoukjun Kwon, Michael Pellauer, Angshuman Parashar, and Tushar Krishna. 2022. A Formalism of DNN Accelerator Flexibility. *Proc. ACM Meas. Anal. Comput. Syst.* 6, 2, Article 41 (June 2022), 23 pages. https://doi.org/10.1145/3530907

1 INTRODUCTION

Machine learning (ML), especially deep learning (DL), has demonstrated great results in computer vision, natural language processing, and recommendation systems [9, 17, 28, 33]. This has energized the field of computer architecture to develop customized hardware accelerator ASICs to enable deployment (i.e., inference) of DL solutions in disparate environments like the edge and cloud. Once deployed, the overall runtime and energy-efficiency of running a DNN on the accelerator depends on its *mapping* [30, 36], which determines how tiles of computation are staged on the accelerator across space and time. Traditionally, the efficiency of domain-specific accelerator ASICs has come from *specialization*, i.e., the control path and datapath in the accelerator are tailored to

Authors' addresses: Sheng-Chun Kao, Georgia Institute of Technology, USA, felix@gatech.edu; Hyoukjun Kwon, Georgia Institute of Technology, USA, hyoukjun@gatech.edu; Michael Pellauer, NVIDIA, USA, mpellauer@nvidia.com; Angshuman Parashar, NVIDIA, USA, aparashar@nvidia.com; Tushar Krishna, Georgia Institute of Technology, USA, tushar@ece.gatech. edu.



This work is licensed under a Creative Commons Attribution International 4.0 License.

© 2022 Copyright held by the owner/author(s). 2476-1249/2022/6-ART41 https://doi.org/10.1145/3530907 the deep neural networks (DNNs) that are expected to run on the accelerator. In other words, the number of mappings that an accelerator can support (aka *map-space*) is restricted.

Specialization is unfortunately a double-edged sword. While high-specialization enables DNN accelerator ASICs to provide higher performance and better energy-efficiency than CPUs and GPUs, it restricts the accelerator from adapting to the growing diversity in DNN models today. This is becoming a key challenge, given how rapidly the field of ML is evolving, since it is highly probable that accelerator chips will be obsolete by the time they reach the market.

To this end, there has been growing interest in developing *flexible* DNN accelerators. In contrast with a fully-programmable (i.e., Turing Complete) processor such as a CPU/GPU, or a fullyreconfigurable circuit such as a FPGA, a flexible accelerator adds small-overhead-but-high impact reconfigurability. The flexibility manifests in enabling the DNN accelerator to support diverse mapping strategies (i.e., larger map-spaces) for the same DNN layer. Intuitively, this works since diverse DNN layers can be thought of as different shapes and sizes of a few key fundamental operators such as convolutions or matrix-multiplications; the ability to change between mappings at runtime after silicon deployment allows the compiler to map skewed dimensions of different operations over space or time to enhance utilization and hide memory access times behind computations.

Flexibility provides a two-fold benefit. First, it allows the accelerator to better tailor itself to the diverse set of layer parameters within the current DNN being mapped [30, 36], instead of targeting the average case. For example, the CONV2_1 early layer in ResNet-50 has 56x56 activations with 64 channels, whereas the CONV5_3 late layer in ResNet-50 has 7x7 activations with 2048 channels. An accelerator that only supports parallelism on activations can result in severe under-utilization on the late layer, and vice versa for one that only supports parallelism on channels. Second, it helps "future-proof" the accelerator. For instance, consider an accelerator such as Eyeriss [12], developed in 2014 for accelerating AlexNet [29], the state-of-the-art network at that time for image classification, which had 3x3, 5x5 and 11x11 filters; Eyeriss is severely under-utilized for newer DNN models that use layers such as pointwise (i.e., 1x1 filters) convolutions (e.g., MobileNetV2 [40]), as demonstrated by its successor Eyeriss_v2 [13], or attention (e.g., BERT [15]).

While the notion of flexibility described above makes intuitive sense, the field of DNN accelerators today is inconsistent about the definition. We find that "flexibility" has been used loosely to refer either to the ability to handle different loop tiling/blocking [45], and/or support for different loop orders [32], and/or the ability to spatially partition across different dimensions [13], and/or the ability to logically support different PE aspect ratios [31]. In fact, there is also inconsistency in whether flexibility is a hardware feature [30] or simply a term for compiler loop-transformations over a fixed inner tile [45, 46]. This in-formalism is a serious barrier to the adoption of flexibility features, as it hinders precise quantification of the cost/benefit tradeoff. Moreover, addition of large map-spaces confounds the accelerator Design-Space Exploration (DSE) problem as concepts such as *partial* flexibility cannot be precisely expressed or numerically assessed by existing DSE flows.

In this work, we present a novel formal definition and precise quantification of accelerator flexibility. We also quantify the corresponding cost for flexibility, including area and energy. We then couple that with an abstraction that allows its incorporation into a first-of-its-kind *flexibility-aware* automated DSE, without making the problem intractable. As a debut work in this field, we cannot claim to be exhaustive. However, we advocate that our formalism can be the first step toward the accelerator community moving towards consensus on the costs and benefits of flexible accelerators. Specifically, we make the following contributions:

Contribution 1: We present a formal definition of the flexibility of an accelerator that refers to the percentage of explorable / supported map-space out of all possible (i.e., exhaustive, as formally defined in Section 4) mappings for a DNN layer. We distill this down further into *workload-dependent*

(i.e., what a compiler/ mapping optimizer is legally allowed to explore) and a workload-independent but *hardware-dependent* (i.e., what the hardware is designed to support) map-spaces.

Contribution 2: As these map-spaces can be overwhelmingly complex, we identify a complementary abstraction: we taxonomize flexibility into four axes – tile size (T), order (O), parallelism (P) and array shape (S). Identifying each axis with a binary 0/1 value enables us to create 16 flexibility classes within which various accelerators can fall, enabling a systematic classification of prior work. For each axis, we identify both the hardware support needed, and the software loop-transformations it exposes to a mapper.

Contribution 3: Together Contributions 1 and 2 allow us to precisely specify constraints on the shape of the map-space in order to distinguish diverse accelerators with *varying degrees of flexibility* that we quantify with a new metric called *flexibility fraction* or *flexion*¹ during automated DSE. We demonstrate flexibility-aware DSE using a genetic algorithm, constrained by the amount of flexibility provided by a target accelerator along each of the four axes.

Contribution 4: We leverage the above contributions to perform isolation analysis, comparing several accelerator design points along each axis (independently and in conjunction), identifying the cost and benefit of flexibility for modern DNNs across vision, language and recommendation. We also evaluate a first-of-its-kind "what-if" scenario to determine how a vision DNN accelerator designed in 2014 would have fared on DNN models today if it had been made future-proof by adding flexibility across one or more axes. We summarize our key takeaways:

- We find that making a 2014 DNN accelerator future-proof by increasing flexibility can significantly improve runtime on future (i.e., present-day) DNNs (11.8× geomean).
- For Mnasnet, independently, (T) and (P) yield the most (4.8x/1.7x) performance gain and (S) the least (1.04x).
- "Partially" flexible accelerators easier to build and program can sometimes provide similar benefits to fully-flexible ones (12-57% depending on the network).
- For DNN models formed with similar set of layers, (T) offered the most valuable flexibility (3.2x speedup). For diverse models, accelerators providing flexibility along (P) and (S) together offered higher benefits in runtime (49%) compared to investing in only one of them (8-18%).
- Area overheads for flexibility are low (<1%) in practice. Interestingly, we find no energy overhead for this area increase, as the features pay for themselves via better mappings that reduce DRAM accesses.

In all, we hope that quantifying the benefits of flexibility can help the community better understand the value of investing engineering effort into flexible hardware accelerators, and the software mapper toolchains required to exploit it.

2 BACKGROUND

2.1 DNN Accelerator Components

Fig. 1 shows an example of a DNN accelerator. It is comprised of a spatial array of processing elements (PEs), buffers (across a memory hierarchy), and networks-on-chip (NoC) for operand distribution and output reduction/collection. In Section 3.4, we discuss the hardware support needed in each of these axes to provide "flexibility".

2.2 Mappings and Dataflows

Each layer of a DNN can be expressed as a multi-dimensional loop nest over the input and weight tensors. A CONV2D layer has 6 loops (7 with batching) while a FC layer (i.e., GEMM operation)

¹Pronounced "fleck-shun." In biology this term refers to the act of bending a joint, making it apt for a flexibility metric.



Fig. 1. An example NVDLA mapping (K/C: Output/input channels. Y/X and R/S: height/width of activation and weight).

has 3 loops. At runtime, these loop bounds may be smaller than the hardware resources available (leading to under-utilization) or greater than them (leading to multiple temporal passes). Scheduling the loop involves splitting these loops into sub-partitions (tiles) and re-ordering the resulting loop nest. Different accelerators choose to execute different loop levels temporally (via serialization through the same PE) or spatially (via parallelization across distinct PEs, or clusters of PEs). The choice of parallelization may allow some input tensors to be multicasted to multiple PEs, amortizing expensive accesses. Similarly, parallelized partial-sum contributing to the same output allows for the use of efficient spatial reduction hardware. The number of tiling levels, loop order, and parallelism dimensions are collectively referred to as the accelerator's *dataflow* in prior work [11, 30]. The dataflow, with specific tile sizes at each level of the accelerator's buffer hierarchy, is called a *mapping* [30, 36]. Fig. 1 shows a loop-nest example for the NVDLA mapping [35]. We dissect it further in Section 3.1.

2.3 Map-Space Exploration (MSE)

A given mapping's efficiency is determined by the utilization it achieves and the total number of buffer accesses it makes—particularly to expensive off-chip memories. Prior works have demonstrated that different DNN layers prefer different mappings [30, 36]. This has motivated research in tools called *mappers* to find optimized mappings given a layer and hardware target. The search-space of mappings that can run on an accelerator (aka *map-space*) is often extremely large ($O(10^{24})$ [27]), so these mappers rely on heuristics [8, 36], optimization methods [24, 27] or ML-based surrogate models [22] for faster searches.

This notion of a map-space includes all possible loop transformations and tile sizes for a layer that a Turing-Complete computer could execute. One additional challenge with domain-specialized accelerators is that not all mappings are *valid* given the target hardware's concrete parameterized resource constraints such as size of buffers, bandwidth, number of PEs, and so on. Just as neural networks tend to *over-fit* to their training data, many DNN accelerators employ hard-coded control and datapaths in order to increase efficiency on known workloads [11] and to keep the hardware simple, which limits the choices of dataflows/mappings that can run on it. These *partially* flexible accelerators actually pose stricter constraints to the loop transformations and tiling than the size of HW resources do. For example, a mapping tool targeting a 32×32 spatial array has more in common with the same array at size 128×128 with larger buffers, than it does with a variant that supports spatial reduction of sums.

2.4 Design-Space Exploration (DSE) Challenges for Flexible Accelerators

Traditional DSE [30, 36] for inflexible accelerators is similar to MSE: each point in the map-space reflects a potential design point that can be "hardened" into silicon that is specialized to do exactly

that mapping of operations, with a particular provisioning of buffering, computation, and on-chip network bandwidth. In this inflexible scenario, the design-space can be explored using the same techniques and heuristics (or even the same tool) as the map-space.

Making the candidate accelerators themselves flexible means that each of the millions of potential candidate accelerator designs are now individually associated with a particular map-space that is derived from whatever hardware features it employs—e.g., flexible buffers, flexible address generation, flexible operand-delivery networks, etc (see Section 3.4). The contents of these map-spaces can range from a few significantly different mappings, to septillions [27] of configurations with subtle differences. This means that every step of a DSE *includes multiple MSE*, because the DSE needs to credit the design with its best-possible mapping for every benchmark DNN layer.

Rather than trying to solve this difficult co-optimization problem, we formalize flexibility through an abstracted taxonomy (Section 3) and precise quantification (Section 4). Together, these allow practical integration of flexibility into the DSE process.

3 AXES OF ACCELERATOR FLEXIBILITY

We define accelerator flexibility as "how many different ways can we compute this workload." In other words, flexibility aims to describe a device's ability to run different mappings (i.e., re-arrangements or transformations) of the same starting loop nest, as opposed to unrelated loop nests from different programs (which might be a measure of programmability).

We follow a bottom-up approach towards understanding flexibility. In this section, we identify axes of flexibility and qualitatively discuss the performance benefit and HW cost for supporting each. In Section 4, we identify the map-spaces exposed by each axis. In Section 5, we discuss our mechanism to search through these flexibility-constrained map-spaces.

3.1 Axes Taxonomy

Informed from existing work of accelerators [2, 13, 16, 18, 19, 25, 35, 38, 39] and map space exploration tools (MAESTRO [2], Timeloop [36], TVM [10], and so on [14, 45]), where intra-layer mappings are captured as loop nests, we identified and unified the knobs that these tools vary when doing map-space exploration. We distill flexibility into four axes².

(1) Tile sizes (T): The ability to change bounds and aspect ratios of data tiles from one or more operand tensors per level of the buffer hierarchy [37].

(2) Loop order (O): The ability to change the loop orders iterated per tiling level.

(3) Loop parallelization (P): The ability to change which tensor dimensions are parallelized per tiling level. This represents the *spatial* partitioning of data (i.e., across PEs).

(4) Array shape (S): The ability to change the logical shape and clustering of the accelerator's hardware resources. This determines the number of tiling levels and the maximum tile sizes for the tensor dimensions being mapped in parallel (i.e., spatially) over the accelerator array.

We refer to the four axes as flexibility TOPS (tiling, order, parallelism, shape), as a mnemonic with the well-known performance metric Tera-Operations Per Second. Fig. 1 shows an example of NVDLA mapping [35], where in the 6-order for loop of convolution, tile sizes are (16, 64, 3, 3, 4, 4), loop order is (K, C, R, S, Y, X), loop parallelization is (K, C) parallelism, and array shape is (64x16).

Scope of Classification. Note that the four axes we described exhaustively capture the space of *intra-layer mappings* which has been the target of several recent mapping optimizers [8, 22, 24, 27, 36]. We assume that an accelerator runs each layer sequentially, consistent with state-of-the-art DNN accelerators [12, 16, 25, 31, 35, 39, 41], and flexible accelerators can change the mapping alone one or more of these axes for each layer. Note that inter-layer mapping strategies (e.g., loop

²We only consider intra-layer mapping (which most accelerators target) and did not consider inter-layer mapping.



Fig. 2. (a) Flexibility TOPS and prior work. (b) For any workload, an unique mapping involves choosing individual points from the map-space actually supported by the target accelerator.

fusion [5]) can be a outer-loop on top and not the focus of this study. Further, aspects of DNN accelerators tangentially related to flexibility that we explicitly leave to future work include sparsity and compression, bit-serial arithmetic, alternate algebraic refactorings such as Winograd [44] or UCNN [23], and processing-in-memory and near-memory architectures.

3.2 Classes of Accelerator Flexibility

We introduce a notation to concisely describe an accelerator as a binary vector $[X_T, X_O, X_P, X_S]$ where:

$$X = \begin{cases} 0 \text{ if } |Mappings_{supported}| = 1 \text{ (inflexible)} \\ 1 \text{ if } |Mappings_{supported}| > 1 \text{ (some flexibility)} \end{cases}$$
(1)

Mappings_{supported} is the number of legal mappings for a given workload (i.e., DNN layer) over a given accelerator.

Our taxonomy creates 16 accelerator classes - with Class 0 (i.e., $[X_T, X_O, X_P, X_S] = [0, 0, 0, 0]$) being the least flexible and Class 15 (i.e., [1, 1, 1, 1]) being the most. The intent of this taxonomy is to enable a coarse classification of accelerators and allow a systematic study of cost-benefit analysis. In Fig. 2(a), we do a best-effort classification of several current accelerators into these classes. Fig. 2(b) then demonstrates the role of a compiler/mapper: it generates unique mappings by selecting from the subset of the map-space that is actually supported by the target accelerator [36].

3.3 Benefits of Mapping Flexibility Per Axis

In Fig. 3 we show how each axis of flexibility can enhance performance and/or energy-efficiency depending on the workload dimensions. For simplicity, we use a GEMM mapped on a 4x4 array throughout the example.

In Fig. 3(a), suppose we have a GEMM with (M=8, K=4 and N=8). It is tiled such that the size of A tiles (i.e., input activations) is 16 (M_t =4, K_t =4), and that of B tiles (i.e., weights) is 32 (N_t =8, K_t =4) - assuming this constraint comes from the sizes of the input and weight buffers around the array. The 32 weights will require two iterations or passes through the 4x4 PE array during which time 16 of the weights will remain stationary; the two sources of inefficiency are (i) the same inputs will be read from the global buffer and streamed during each iteration (increasing energy), and (ii) switching the stationary tile will lead to a stall. Now consider a flexible accelerator, where the



Fig. 3. The benefit of having flexibility in (a) Tile, (b) Order, (c) Parallelism, and (d) Shape. Example GEMM operation of $Z_{MN} = A_{MK} \times B_{KN}$ on 4x4 PE array. The subscript *t* indicates that these are tiles of the overall computation. The inflexible accelerator is weight stationary (tensor *B*), using a loop ordering of $K \to N \to M$.

global buffer is soft-partitioned and can be configured to store a tile of 32 inputs (M_t =8, K_t =4) and 16 weights (N_t =4, K_t =4) instead. Now, the 16 weights will remain stationary as the 32 inputs stream.



Fig. 4. HW Overhead for Flexibility Support (highlighted in yellow). (a) Flexible Tile Size HW Support: Soft-partitioned Scratchpads. (b) Flexible Loop Order HW Support: Configurable Address Generators. (c) Flexible Parallelism HW Support: Spatial and Temporal Reduction. (d) Flexible Shape HW Support: NoC support for PE clustering.

Here only one iteration (although longer) will be needed, only one L2 buffer read for each input required and no stall to switch stationary tiles, potentially reducing energy and runtime.

In Fig. 3(b), the A tile has size 16 while the B tile has size 32. Keeping the B tile stationary requires two iterations and two reads of the A tile, leading to a similar issue as Fig. 3(a). In contrast, if we could switch the loop order to keep the input tile stationary instead as the flexible accelerator does, we can finish the computation in one iteration with fewer buffer reads, enhancing performance and energy-efficiency.

In Fig. 3(c), suppose the GEMM is not square, but rectangular. In the default mapping discussed so far, the parallelism is across the K and N dimensions which leads to a utilization of only 50%. Here the K dimension is reduced spatially across the columns (like the TPU [25]). If the accelerator could support flexible parallel dimensions, then we could switch to MN parallelism (i.e., output-stationary dataflow), performing the reduction temporally within each PE. Now each PE operates on a separate output and we get 100% utilization.

In Fig. 3(d), we see that the dimensions of the weight tile (2×8) do not match those of the 4x4 array, leading to two iterations, each with 50% utilization. Flexibility in the shape of the physical array would enable it to emulate a logical 2×8 aspect ratio instead, increasing utilization to 100%.

As shown in Fig. 3, different axes of flexibility provide different opportunities to enhance utilization and/or reuse. In fact, in some cases more than one choice exists for enhancing the utilization. E.g., the benefit achieved by flexible shape for the tall-skinny GEMM in Fig. 3(d) could also be achieved by switching to MN parallelism like Fig. 3(c). This motivates this paper's focus on dissecting the performance and energy benefits along each axis of flexibility.

3.4 Hardware Support for Flexibility Per Axis

However the flexibility is not free. It comes with an area cost, as we characterize later in Table 3. For example, a multi-casting circuit takes lager area than a uni-cast; flexible choices of inputs introduce multiplexers whose area cost grows with number of choices. We characterize the specific hardware cost of each axis of flexibility via Fig. 4 as follows.



(a) Map Space (b) H-F = A_X/C_X (c) W-F = $A_X^{\omega}/W_X^{\omega}$ (d) Full/Part/In-Flex Fig. 5. (a) Venn diagram of map space of a class-X target accelerator. (b) Hardware-dependent Flexion. (c) Workload-dependent Flexion. (d)A 2D example of the relationship between flexibility axes, flexibility class, and degree of flexibility. Definitions are in Table 1.

Tile: To support tile flexibility, the accelerator needs to be able to access different tiles in the buffer. As shown in Fig. 4(a), this costs additional Base (memory start point), Bound (data length), and Current (the current address pointer) Registers for input, weights, and outputs buffers, which are controlling/monitoring the data flow of accessing current tiles. In addition, to be able to support more varieties of tile shapes, the accelerator should have a software-controlled soft-partitioned buffer (instead of hard-partitions across each operand) to enable more tiling opportunities, shown in Fig. 4(a) via a multiplexer/demultiplexer.

Order: To support different loop orderings, the accelerator needs to allow tiles from different tensors being stationary versus streaming. It needs to have additional address counters and address generators for inputs, weights, and outputs, as shown in Fig. 4(b). Each PE also needs a register to store the counter value it should count up to before discarding the stationary tile.

Parallelism: To support different parallelism dimensions, we need three additional address counters and address generators. In addition, we need a dedicated multiplexer in front of each PE to select between a spatial or temporal reduction path, as shown in Fig. 4(c).

Shape: Shape flexibility requires the ability for the accelerator to mimic (i) a larger row/column by sending the same operand across multiple rows/columns and/or (ii) a smaller row/column by breaking into multiple parts. E.g., in Fig. 3(d), the flexible accelerator runs two spatial reductions of size two on the same column. Feature (i) needs a richer distribution NoC topology to multicast the operand across multiple rows/columns. Feature (ii) requires demuxes at the output of each PE to allow it to forward the output to the neighbor or directly to the L2 buffer via a reduction NoC that provides these connections, as Fig. 4(d) shows. Prior work [30, 31] has discussed implementation choices for such NoCs.

We wish to highlight that beyond the area and power cost incurred by the components described above (which we find in our evaluations to not be very significant over the MAC, buffer, and NoC required by an inflexible accelerator), a key cost for adding flexibility support is that of *engineering and verification*. This is why a systematic understanding of the benefits of flexibility is valuable for DNN accelerator developers.

4 FORMAL FLEXIBILITY-CONSTRAINED MAP SPACE DEFINITION

Given the above insights, we now formalize how hardware flexibility and workload specialization precisely affect the feasible map space.

4.1 Definitions of Map Space and Flexibility

Table 1 summarizes the key definitions. We omit the mathematical equations in the interest of space, instead use the Venn diagrams in Fig. 5 to guide the discussion.

Target accelerator: The accelerator whose map-space is being evaluated.

C_X	Map space of an accelerator class (class-X)
A_X	Map space of the target accelerator in class-X
W_X^{ω}	Map space of workload ω
C_X^{ω}	$C_X \cap W_X^{\omega}$. Workload-dependent map space of class-X
Δω	$A_X \cap W_X^{\omega}$. Feasible map space. Workload-dependent
A_X	map space of the target accelerator
	A_X/C_X . Hardware-dependent Flexion of the target
H-F	accelerator. Supported map space percentage
	in the entire class-X map space.
	$A_X^{\omega}/W_X^{\omega}$. Workload-dependent Flexion of the
W-F	target accelerator. Supported map space percentage
	in the entire workload ω space.

Table 1.	Definition	of Map	Space and	Flexibility.
----------	------------	--------	-----------	--------------

Workload: Within the scope of our evaluations, this refers to a given layer of a given DNN model, since we consider the accelerators mapping layers one by one. The venn diagrams and definitions from this section will however also hold for workloads referring to full models or fused layers.

Mapping: A design-point in the map space, which precisely describes the value for T, O, P, and S.

Class-X: We use class-X to denote the class of the target accelerator (Section 3.2). Note that accelerators with different configurations of HW resources such as number of PEs, buffers, and NoCs are all called class-X accelerators as long as their supporting flexibility axes are categorized into X class. Note that the following definitions applies to all of the 16 accelerator classes, i.e., we can use it to discuss map-space and flexibility of class-0011 with two axes enabled, class-1111 with four axes enabled, and so on.

Map space: the design space of the mapping of a class-X accelerator. E.g. the map space of class-0011 will expand along the two flexible axes P and S, while T and O are fixed values across all mappings. These fixed values are often practically hard-coded at accelerator design-time for simplifying hardware design (Section 3.4).

Hardware-dependent Map Space: We define two hardware-dependent map-spaces: C_X and A_X .

 C_X is used to denote the map space of a class-X accelerator given a constraint in total HW resources available. Therefore, C_X is a constrained map space. Taking Class-1xxx accelerators as an example, the total buffer size within the accelerator will directly limit the number of possible tiling choices. Similarly, for Class-xxx1, the total number of PEs are the constraint. No such resource constraints exist for order and parallelism. C_X can be viewed as the map-space of a fully-flexible accelerator within that class. Note that C_X is *workload-agnostic*. For e.g., C_{1000} captures all possible tile sizes that can fit in the buffers, agnostic of tile sizes that make sense for the workload (as we discuss later in this section). Thus in Fig. 5, it captures a map-space beyond what the specific workload needs.

Given the same HW resource constraint, the target accelerator in class-X might add *additional constraints*, creating its own map space A_X . A_X will always strictly be a subset of C_X , as shown in Fig. 5. For e.g., given a total buffer size, an accelerator with hard partitioned buffers for inputs, weights and outputs will cover a smaller map-space (i.e., A_X) than theoretically possible in one that provides full flexibility in partitioning the buffers (i.e., C_X).

Workload Map Space: This denotes the possible mappings for a given workload ω regardless of underlying HW, noted as (W_X^{ω}) . For example, for a CONV layer in ResNet50 (CONV2_1), with 64 output and 64 input channels, 3x3 weight kernel, and 56x56 activations, the entire workload map



Fig. 6. Flexibility-Aware Automated Design-Space Exploration Framework.

space is all the possible combinations of 6 parameters with values from 1 to its dimension size (e.g., 1-64 for output channel). However, not every tiles can fit into the limited on-chip buffer (whose map space is captured by C_X). Therefore, for C-1xxx class, C_X often cannot cover the full W_X^{ω} , as Fig. 5 shows.

Workload-dependent Map Space: This is the intersection of hardware-dependent map space $(C_X \text{ and } A_X)$ and the workload map space (W_X^{ω}) , which creates C_X^{ω} and A_X^{ω} in Fig. 5.

Feasible Map Space: A_X^{ω} , the map space of the target accelerator on a given workload, is the actual map space the mapper is allowed to explore. The size of A_X^{ω} is the value of $|Mappings_{supported}|$ in Equation (1).

Flexibility Fraction or Flexion: The fraction of supported map space to the possible map space. *Hardware-dependent Flexion (H-F)*: Accelerators in the same flexibility class can be implemented with different degree of flexibility, whose feasible map space percentage is A_X/C_X . We call this hardware-dependent Flexion (H-F).

Workload-dependent Flexion (W-F): Different workloads will have different shape and size of the map space. To understand how much portion of the map space the target accelerator can support for the current workload, we define workload-dependent Flexion (W-F) as $A_X^{\omega}/W_X^{\omega}$.

4.2 Full and Partial Flexibility

Any accelerator in class-X can have a grey-scaled degree of flexibility support (0< flexibility degree <=1). This is illustrated for two axes of flexibility (for simplicity) in Fig. 5(d). For ease of referring, we use the terms FullFlex-X, PartFlex-X and InFlex-X in our evaluations in Section 6 and Section 7. For e.g., FullFlex-0001 is an accelerator that supports all possible shapes S (such as MAERI [31]), while PartFlex-0001 is an accelerator that supports a subset of shapes S (e.g., Eyeriss [12] to simplify hardware), and InFlex-0001 supports no flexibility in shape (such as the 128x128 systolic arrray within TPU-v3 [25])³.

5 FLEXIBILITY-AWARE DESIGN-SPACE EXPLORATION

Ideally, the above formalism is (A) precise enough to capture a quantifiable constraint on the shape of the map-space, and (B) abstract enough not to make the DSE problem intractable. To demonstrate this, we extend an open-source DNN mapper, GAMMA [1, 27], which uses a genetic algorithm to search through a large search-space ($\sim 10^{24}$ [27]) in a sample-efficient manner. We use the same genetic search technique for DSE and MSE where applicable.

Modules for Flexibility-awareness. The native mapper only supports either inflexible accelerators (accelerators in class-0000 aka InFlex-0000) or fully flexible accelerator in class-1111 (FullFlex-1111). We extend it with two additional features: (i) Ability to constrain the search within 16 different accelerator classes (i.e., FullFlex-xxxx). (ii) Ability to constrain the search further in each class for PartFlex accelerators.

³InFlex-0001 is equivalent to InFlex-0000 but we make the S-bit high for ease of comparison against the partial and fully flexible versions in that class.

	are resources and baseline riexibility rors:				
HW Resources	Number of PEs: 1024, on-chip buffer: 100KB				
Mapping Config	T: [K:64,C:16,Y:3,X:3,R:3,S:3] tile size,				
	O: KCYXRS order, P: K-C parallel, S: 16x64 PE array				

Tabl	e 2.	Hard	ware	resources	and	base	line	flexi	bil	lity	' TOP	S.
------	------	------	------	-----------	-----	------	------	-------	-----	------	-------	----

Table 3. Area cost of accelerators with different flexibility.

	InFlex	T-Flex	O-Flex	P-Flex	S-Flex	PartFlex	FullFlex
Area	736 843	763,874	738,458	737,720	737,055	738,209	739,576
μm^2	/ 30,043	(+0.004%)	(+0.21%)	(+0.11%)	(+0.02%)	(+0.19%)	(+0.37%)

These features are implemented by adding different levels of constraints to the map space of the native mapper. It includes modules for understanding the flexibility specification from input, encoding flexibility into constraints and updating the mapping exploration operators [27] correspondingly⁴.

Modules for Area, Power for Estimating Cost of Flexibility. To estimate hardware overhead, we implemented RTL of the various components described in Fig. 4, synthesized them using Synopsys DC with Nangate 15nm library [3] and used Cadence Innovus for place-and-route. We synthesized the SRAM buffers with SAED32 education library from Synopsys, and scaled it to 15nm. The final cost model takes in the mapping constraint file to understand the required flexibility of the accelerator and outputs its area/power/energy cost. We modularized these building block and build a cost model, which is embedded in flexibility-aware DSE (Flexibility overhead cost estimator in Fig. 6).

Toolflow. As shown in Fig. 6, the flexibility-aware DSE takes in DNN model description, baseline HW resources, and HW flexibility specification. Three of them together defines (or so-called selects) the map space that the internal MSE tool works on. After its internal MSE tool converges or reaches a pre-defined number of sampling budget⁵, it terminates the optimization and outputs the best-found design point and its HW performance (runtime, energy, area, and power).

6 EVALUATION I: ISOLATION STUDY ON FLEXIBILITY AXES

In this section, we demonstrate how architects could utilize the flexibility-aware toolflow to tractably explore different map-space constraints on candidate designs. Simultaneously, we leverage the toolflow constraints to target the DSE in order to isolate and quantify the contributions of the T/O/P/S axes.

6.1 Methodology

Workloads. We use MnasNet [43], which is a high speed and high accuracy model found by Neural Architecture Search (NAS), for our evaluation, since networks found by NAS have been recently achieving state-of-the-art performance.We also look into three types of models: Vision (Alexnet [29], Resnet50 [20], MobilenetV2 [40]), Language (BERT [15]), and Recommendation (NCF [21], DLRM [34]).

Target Accelerators. For the targeted isolation studies, we formulate three accelerators with different levels of flexibility: i) an inflexible accelerator (**InFlex**), using a fixed and unchangeable TOPS configuration, ii) a partially flexible accelerator (**PartFlex**) with some constrained capabilities that we describe in the relevant sections, and iii) a fully flexible accelerator (**FullFlex**), which can support the full map-space, as was described earlier in Section 4.2. These accelerators are built with

⁴We do not go into engineering details in the interest of space, but will open-source the mapper upon paper acceptance.

⁵Throughout the experiments, we set the genetic algorithm hyper-parameter as 100 populations, 100 generations, and thus having 10K sampling budget. The mutation/crossover rate and execution rate of other evolving functions as 0.5. We set these by applying a hyper-parameter search [7].



Fig. 7. Tile Axis Isolation. (a) Performance comparisons of accelerators with different level of Tile flexibility running Mnasnet model. (b) Venn diagram of Tile map space on Mnasnet. We use onchip buffer size=4K in this experiments. Scale: Total Data points in $W_T^{\omega} = \pi (40)^2$. Runtime/Energy/EDP are normalized against values of InFlex-1000.



Fig. 8. The runtime performance and workload Flexion of fully-Tile-flexible accelerators with different buffer sizes. (W-F: Workload-dependent Flexion.)

the HW resources and baseline configuration listed in Table 2, and the area overhead for different flexibility is shown in Table 3. For each study, we characterized the full neural network across all layers and present the overall runtime, energy, and energy-delay product (EDP) for the best-found mappings. Furthermore, we then present detailed analysis of three layers with significantly different aspect ratios from MnasNet to illustrate the correlation between performance.

Optimization Objective. Across our experiments, we set the optimization objective to minimum runtime. However, other objectives such as energy, area, energy-delay-product (EDP), performanceper-watt, and so on, are all feasible. We can also use multi-objective formulation to optimize two or more objectives simultaneously.

PE=16x64	Accel	Runtime	Energy	EDP	H-F	W-F	Chosen
I1(InFlex0100	1.00	1.00	1.00	0.001	0.04	YXKCRS
Layer16	PartFlex0100	0.82	1.003	0.82	0.004	0.13	KCRSYX
(120,40,28	FullFlex0100	0.80	0.9997	<u>0.80</u>	1.00	1.00	CXYKRS
,20,1,1)	FullFlex1111	Runtime Energy EDP H-F W-F Ch (0100) 1.00 1.00 1.00 0.001 0.04 YX $(x0100)$ 0.82 1.003 0.82 0.004 0.13 KC $(x0100)$ 0.80 0.9997 0.80 1.00 1.00 CX $(x111)$ 0.002 0.99 0.002 - - YC (0100) 1.00 1.00 1.00 0.004 YX $(x0100)$ 0.002 0.99 0.002 - - YC (0100) 1.00 1.00 1.00 0.004 0.13 KC $(x0100)$ 0.77 0.998 0.77 0.004 0.13 KC $(x0100)$ 0.74 0.997 0.74 1.00 1.00 CK $(x0100)$ 1.00 1.00 1.00 0.001 0.01 YX $(x0100)$ 1.00 1.00 0.001 0.01 YX <t< td=""><td>YCKSXR</td></t<>	YCKSXR				
T 01	InFlex0100	1.00	1.00	1.00	0.001	0.04	YXKCRS
Layer21	PartFlex0100	0.77	0.998	0.77	0.004	0.13	KCRSYX
J.2011111FullFlexLayer21InFlex0(40,120,28PartFlex0(40,120,28FullFlex0J.282,1,1)FullFlex0Layer29PartFlex0(1,480,14,FullFlex014, 5,5)FullFlex0Mnasnet-PartFlex0PartFlex0PartFlex0	FullFlex0100	0.74	0.997	<u>0.74</u>	1.00	1.00	CKXYRS
	FullFlex1111	0.002	1.29	0.002	-	-	KCYRXS
I	InFlex0100	1.00	1.00	1.00	0.001	0.01	YXKCRS
Layer29	PartFlex0100	1.00	1.09	1.09	0.004	0.03	KCRSYX
(1,400,14, 14 5 5)	FullFlex0100	0.997	1.001	<u>0.998</u>	1.00	1.00	KCXYRS
,28,1,1) Layer21 (40,120,28 ,28,1,1) Layer29 (1,480,14, 14, 5,5) Mnasnet- Model InFlex02	FullFlex1111	0.0003	0.48	0.0001	-	-	KSYRXC
	InFlex0100	1.00	1.00	1.00	0.001	0.01	-
Layer16 InFlex0100 1.00 1.00 1.00 0.001 0.001 PartFlex0100 0.82 1.003 0.82 0.004 0.1 FullFlex0100 0.80 0.9997 0.80 1.00 1.00 FullFlex0100 0.80 0.9997 0.80 1.00 1.00 Layer21 InFlex0100 1.00 1.00 1.00 0.001 0.001 Layer21 InFlex0100 0.77 0.998 0.77 0.004 0.1 Layer23 InFlex0100 0.74 0.997 0.74 1.00 1.00 FullFlex1111 0.002 1.29 0.002 - - Layer29 InFlex0100 1.00 1.00 1.00 0.001 0.0 Layer29 InFlex0100 1.00 1.00 1.00 0.001 0.0 Layer29 InFlex0100 1.00 1.00 1.00 0.001 0.0 FullFlex1111 0.002 1.00 1.00 0.001 <td>0.03</td> <td>-</td>	0.03	-					
Model	FullFlex0100	0.89	0.996	<u>0.89</u>	1.00	1.00	-
	FullFlex1111	0.004	1.00	0.004	-	-	-
InFlex0100 PartFlex0100 ^(a) FullFlex0100							
C_X		AX	C_X		Av	S _X	A _X

Fig. 9. Order Axis Isolation. (a) Performance comparisons of accelerators with different level of Tile flexibility running Mnasnet model. (b) Venn diagram of Tile map space on Mnasnet. Runtime/Energy/EDP are normalized against values of InFlex-0100.

X=0100

Map-space Visualization. Finally, for each of the targeted isolation studies, we present a Venn diagram in the style of Fig. 5 drawn to scale (provided in each figure) with the areas capturing the number of mappings. Each Venn diagram plots the average of the metrics across all layers of the model.

Axis Isolation. We investigate the impact of standalone flexibility of Tile (class-1000), Order (class-0100), Parallelism (class-0010) and Shape (class-0001) with the target accelerators described by varying the constraints given to the tool. This aspect of the evaluation is not intended to reflect the typical usage of the tool, but rather as a limit study that characterizes the most extreme points in the overall design space.

6.2 Isolation: Tile Flexibility

We study the impact of tile flexibility with three kinds of accelerators: InFlex-1000 (or InFlex-0000⁶), uses the fixed baseline tile sizes; PartFlex-1000 uses hard-partitioned buffer with partition ratio of 1:1:1 for weight, input, and output buffer, and uses flexible tile sizes; FullFlex-1000 uses soft-partitioned buffer and flexible tile sizes. Note that the hardware trade-off between hard-partitioning and soft-partitioning significantly increases the energy per on-chip memory access, but can also result in fewer expensive off-chip accesses [36].

Map Space. Hardware-dependent Flexion (H-F) A_X/C_X is decided by the partitioning strategy which is around 0.22 for 1:1:1 partition. Value 0.22 is the area ratio of A_X over C_X in Fig. 7(b). A_X^{ω} is

⁶InFlex-X (X=0000 to 1111) indicate the same constraint in all 16 classes.

PE=16x64	Accel	Runtime	Energy	EDP	H-F	W-F	Chosen				
Layer10	InFlex0010	1.00	1.00	1.00	0.03	0.08	KC				
Layer10	PartFlex0010	0.48	1.59	0.76	0.07	0.17	YX				
(72,24,30,	FullFlex0010	0.46	1.20	<u>0.55</u>	1.00	1.00	XK				
50,1,1)	FullFlex1111	0.00003	0.90	0.00002	-	W-F Ch 0.08 F 0.17 Y 1.00 Z - X 0.08 F 0.17 Y 1.00 F 0.17 Y 1.00 F 0.05 F 0.10 Y 1.00 F 0.10 Y 1.00 F 0.07 F 0.14 F 1.00 F 0.10 F 0.14 F 0.05 F	XK				
Layer16 (120,40,28 , 28,1,1)	InFlex0010	1.00	1.00	1.00	0.03	0.08	KC				
	PartFlex0010	0.88	1.85	1.63	0.07	0.17	YX				
	FullFlex0010	0.48	1.50	<u>0.72</u>	1.00	1.00	KS				
	FullFlex1111	0.00006	0.92	0.00006	-	-	KC				
I	InFlex0010	1.00	1.00	1.00	0.03	0.05	KC				
Layer29	PartFlex0010	0.50	0.50	0.25	0.07	0.10	YX				
(1,400,14, 14,5,5)	FullFlex0010	0.05	1.62	<u>0.09</u>	1.00	1.00	RS				
14,5,5)	FullFlex1111	0.0001	0.16	0.00001	SDP H-F W-F Chosen .00 0.03 0.08 KC .076 0.07 0.17 YX 1.55 1.00 1.00 XK 10002 - - XK .00 0.03 0.08 KC .63 0.07 0.17 YX 1.72 1.00 1.00 KS 10006 - - KC .00 0.03 0.05 KC .025 0.07 0.10 YX 1.09 1.00 1.00 RS 00001 - - KY .00 0.03 0.07 - .70 0.07 0.14 - .58 1.00 1.00 - .0002 - - -						
	InFlex0010	1.00	1.00	1.00	0.03	0.07	-				
Mnasnet- Model	PartFlex0010	0.81	0.87	0.70	0.07	0.14	-				
	FullFlex0010	0.59	0.99	0.58	1.00	1.00	-				
	FullFlex1111	0.0004	0.56	0.0002	-	-	-				
	\mathbf{P}_{a}										



Fig. 10. Parallelism Axis Isolation. Performance comparisons of accelerators with different level of Parallelism flexibility running Mnasnet model with (a) 16x64 PE array and (b) 32x32 PE array. (c) Venn diagram of Parallelism map space on Mnasnet. PRuntime/Energy/EDP are normalized against values of InFlex-0010.

the intersection of W_X^{ω} and A_X , showing how many different tiles that workload ω can explore and are supported by the accelerators. A_X^{ω} of InFlex-1000 is 1, since it leverages only 1 choice, while A_X^{ω} of PartFlex-1000 will be smaller than the one for FullFlex-1000, since hard-partitioning is a stricter constraint. These can be observed by the increase of overlapped area of W_X^{ω} and A_X from InFlex-1000 to FullFlex-1000. Note that the workload-dependent Flexion (W-F) $A_X^{\omega}/W_X^{\omega}$ is directly related to the ratio of the supported map space to the entire map space, where larger values implies the given flexibility in the investigating accelerator can support the current workload better.

Performance. PartFlex-1000 allows the tile sizes to be flexible, which reduce runtime by 98%, as shown in Layer-1 (L1) in Fig. 7(a). It chooses smaller K, C sizes and larger Y, X sizes to tailor for Layer-1. While enabling soft-partitioning (FullFlex-1000), with more tile choices, the runtime improves by another 6×. In Layer-16, the C tile sizes are chosen to divide the C dimension size perfectly. The tile configurations are chosen under the criterion of best runtime, and thus energy does not show significant difference. However, if we used least energy as criterion, different points would be picked, demonstrating a larger energy difference between InFlex-1000 and PartFlex-1000/FullFlex-1000. Overall, with this criterion, in class-1000, FullFlex-1000 can achieve speedup of 4.8× and 1.9× over InFlex-1000 and FullFlex-1000 in end-to-end Mnasnet.

Sensitivity Analysis: Buffer Size. The space of supported tile shapes is directly affected by the buffer size, which is the intersection area of S_X and W_S^{ω} in Fig. 7(b). As shown in Fig. 8, with increasing buffer sizes, the workload-dependent Flexion increases and the runtime improves. We can observe that the runtime improvement saturates at a buffer size of around 6.4KB, which is

PE=1024	Accel	Runtime	Energy	EDP	H-F	W-F	Chosen			
	InFlex0001	1.00	1.00	1.00	0.001	0.01	32x32			
Layer15	PartFlex0001A	0.67	1.00	0.67	0.004	0.06	48x16			
ParSize: [72, 40]	PartFlex0001B	0.50	0.97	<u>0.48</u>	0.06	0.25	24x42			
	FullFlex0001	0.50	0.97	<u>0.48</u>	1.00	1.00	24x42			
	FullFlex1111	0.04	0.95	0.03	-	-	72x14			
	InFlex0001	1.00	1.00	1.00	0.001	0.03	32x32			
Layer16	PartFlex0001A	0.75	0.98	0.74	0.004	0.08	16x64			
ParSize:	PartFlex0001B	0.63	0.99	<u>0.62</u>	0.06	0.25	40x24			
[40, 120]	FullFlex0001	0.63	0.99	0.62	1.00	1.00	40x25			
	FullFlex1111	0.04	0.96	0.04	-	-	8x128			
	InFlex0001	1.00	1.00	1.00	0.001	0.01	32x32			
Layer25	PartFlex0001A	0.89	0.99	0.88	0.004	0.05	16x48			
ParSize:	PartFlex0001B	0.89	0.99	0.88	0.06	0.25	16x48			
[80, 480]	FullFlex0001	0.87	0.99	<u>0.86</u>	1.00	1.00	24x42			
	FullFlex1111	0.80	1.21	0.97	-	-	24x42			
	InFlex0001	1.00	1.00	1.00	0.00	0.02	-			
Mnocnot	PartFlex0001A	0.98	1.00	0.97	0.00	0.04	-			
InFlex0001 1.00 1.00 1.00 0.001 0 Layer15 PartFlex0001A 0.67 1.00 0.67 0.004 0 PartSize: PartFlex0001B 0.50 0.97 0.48 0.06 0 FullFlex0001 0.50 0.97 0.48 1.00 1 FullFlex1111 0.04 0.95 0.03 - 0 InFlex0001 1.00 1.00 1.00 0.001 0 Layer16 PartFlex0001A 0.75 0.98 0.74 0.004 0 PartFlex0001B 0.63 0.99 0.62 0.06 0 InFlex0001 0.63 0.99 0.62 1.00 1 PartFlex0001B 0.63 0.99 0.62 1.00 1 Layer25 PartFlex0001A 0.89 0.99 0.88 0.004 0 PartSize: PartFlex0001A 0.89 0.99 0.88 0.004 0 BartSize: PartFlex0001B 0.89 0.99 0.88 0.006 0	0.20	-								
Model	FullFlex0001	0.96	0.99	<u>0.95</u>	1.00	0.98	-			
	FullFlex1111	0.15	1.03	0.15	-	-	-			
			(a)							
InFlex	0001 PartFle	x0001A	Part	Flex000	1B F	uliFlex	001			
$C_{X} \qquad A_{X} \qquad C_{X} \qquad A_{X} \qquad A_{X$										
			(b)	$\overline{)}$	9 (data point	(s) X=0001			

Fig. 11. Shape Axis Isolation. (a) Performance comparisons of accelerators with different level of Tile flexibility running Mnasnet model. (b) Venn diagram of Tile map space on Mnasnet. Runtime/Energy/EDP are normalized against values of InFlex-0001.



Fig. 12. The runtime performance and workload Flexion of fully-Shape-flexible accelerators with different size of PE arrays. (W-F: Workload-dependent Flexion)

sufficient for most of the layers in MnasNet. Conversely, when the buffer size is small, the flexibility to execute on soft-partitioned buffer (FullFlex-1000) becomes more valuable to achieve a good runtime.

Takeaways. For a model like Mnasnet with large diversity in layer dimensions, we find that tile flexibility is crucial for the accelerator to capture the map-space. For the most part, the hard-partitioned PartFlex-1000 provides significant runtime benefits over an InFlex-1000 but is strictly

Proc. ACM Meas. Anal. Comput. Syst., Vol. 6, No. 2, Article 41. Publication date: June 2022.

Flexibility Class	Alexnet	Mnasnet	Resnet50	Mbnet	BERT	DLRM	NCF
InFlex0000-Alexnet-Opt	1.00	1.00	1.00	1.00	1.00	1.00	1.00
InFlex0000-X-Opt	1.00	0.96	0.92	0.96	0.25	0.14	0.07
FullFlex1000-Alexnet-Opt	<u>0.14</u>	<u>0.57</u>	<u>0.07</u>	<u>0.48</u>	0.25	<u>0.99</u>	<u>0.99</u>
FullFlex0100-Alexnet-Opt	1.00	1.00	1.00	1.00	1.00	1.00	1.00
FullFlex0010-Alexnet-Opt	1.00	<u>0.92</u>	1.00	<u>0.88</u>	1.00	0.25	0.57
FullFlex0001-Alexnet-Opt	1.00	<u>0.82</u>	<u>0.91</u>	<u>0.88</u>	1.00	0.58	<u>0.63</u>
FullFlex0011-Alexnet-Opt	1.00	<u>0.51</u>	<u>0.60</u>	<u>0.53</u>	<u>0.90</u>	<u>0.10</u>	0.12
FullFlex0101-Alexnet-Opt	1.00	<u>0.60</u>	<u>0.65</u>	<u>0.61</u>	<u>0.90</u>	0.16	<u>0.40</u>
FullFlex1001-Alexnet-Opt	<u>0.08</u>	<u>0.57</u>	<u>0.07</u>	<u>0.37</u>	0.22	0.32	0.18
FullFlex0110-Alexnet-Opt	1.00	<u>0.60</u>	<u>0.65</u>	<u>0.61</u>	<u>0.90</u>	<u>0.16</u>	<u>0.40</u>
FullFlex1010-Alexnet-Opt	<u>0.06</u>	<u>0.17</u>	<u>0.07</u>	<u>0.15</u>	<u>0.22</u>	<u>0.04</u>	<u>0.04</u>
FullFlex1100-Alexnet-Opt	<u>0.14</u>	<u>0.33</u>	<u>0.07</u>	<u>0.30</u>	<u>0.22</u>	0.66	<u>0.60</u>
FullFlex1110-Alexnet-Opt	<u>0.06</u>	<u>0.17</u>	<u>0.09</u>	<u>0.15</u>	<u>0.22</u>	<u>0.04</u>	<u>0.04</u>
FullFlex1011-Alexnet-Opt	<u>0.06</u>	<u>0.15</u>	<u>0.09</u>	<u>0.14</u>	<u>0.22</u>	<u>0.04</u>	<u>0.04</u>
FullFlex0111-Alexnet-Opt	1.00	<u>0.51</u>	<u>0.60</u>	<u>0.53</u>	<u>0.90</u>	<u>0.10</u>	<u>0.12</u>
FullFlex1101-Alexnet-Opt	<u>0.08</u>	<u>0.33</u>	<u>0.12</u>	<u>0.30</u>	<u>0.22</u>	<u>0.16</u>	<u>0.18</u>
FullFlex1111-Alexnet-Opt	<u>0.06</u>	<u>0.15</u>	<u>0.07</u>	<u>0.14</u>	0.22	<u>0.04</u>	<u>0.04</u>
PartFlex1111-Alexnet-Opt	0.52	0.42	0.20	0.39	0.26	0.07	0.07
Underline: Reach better run	time over	InFlex000	0-Alexnet-	Opt.			
Bold · Reach better runtime	over InFl	-x0000-X-	Ont				

Fig. 13. Runtime performance of accelerators with different flexibility. InFlex-0000-X-Opt means class-0000 (fixed) accelerator that is optimized for DNN model X (X= Alexnet, Mnasnet, etc.). The runtime in each row is normalized by the runtime of the values of InFlex-0000-Alexnet-Opt.

worse than FullFlex-1000. We also observe that tile flexibility is more crucial for accelerators with smaller buffer sizes.

6.3 Isolation: Order Flexibility

The three types of accelerators are configured as follows: InFlex-0100, using the output-stationary order YXKCRS, PartFlex-0100, with three order choices including output, input, and weight stationary order, and FullFlex-0100, with flexible order.

Map Space. H-F A_X/C_X is affected by the number of supported order choices, which depends on the available HW support described in Fig. 4. For order, the map space does not depend on the number of compute or memory resources such as PEs and buffer sizes. Therefore, C_X can encompasses all design points in W_X^{ω} , as shown in Fig. 9(b).

Performance. From Layer-16 in Fig. 9(a), we observe that simply adding 3 out of 6! order choices—and consequently increasing workload Flexion by a small amount—improves runtime to a near-optimal value. This demonstrates that for accelerator design, partially supporting order flexibility may expose a better cost-performance trade-off, if the supported order choices expose distinct behaviors. In our evaluation, we also found that many different orders will end up with the similar runtime. This observation motivates the design of PartFlex–0100, a simpler and lower-cost accelerator that achieves similar performance as FullFlex–0100. Overall, in class-0100, full flexibility support can still achieve 1.12× and 1.01× speedup over the two baselines.

Takeaways. We find that the output stationary loop-order in InFlex-0100 works reasonably well across all layers. Adding support for three loop-orders (output, weight and input) in PartFlex-0100 gets nearly the same performance as supporting all 6! loop orders in FullFlex-0100.

6.4 Isolation: Parallelism Flexibility

The three types of accelerators are as follows: InFlex-0010, using default K-C parallelism, PartFlex-0010, with a choice of K-C or Y-X parallelism, and FullFlex-0010.

Map Space. Since we investigate CONV-accel (6-dim) and consider two-way parallelism. We have C_X =6x5=30 different choices, while the A_X for InFlex-0010 and PartFlex-0010 is 1 and 2, respectively.

Performance. Fig. 10(a) shows that different workloads can have different optimum parallelism choices, and supporting only a subset of choices may not be sufficient to approach optimal performance (Layer-16, 19). Layer-29 is a depth-wise CONV, where there is no cross-input channel computation. InFlex-0010's restriction of parallelism only across K and C dimensions deprives it of other parallelism opportunities such as YX- or RS-parallelism, which could be a better choice for this layer. Overall in class-0010, FullFlex-0010 consistently achieves around 1.6× and 1.3× better runtime relative to the fixed and partially flexible accelerators, respectively.

Takeaways. We observe the flexibility in parallelism is highly-sensitive to the shape of the physical array. We also see that non-conventional parallelism choices (such as XK or KS) are found by the mapper, suggesting that supporting full flexibility in parallelism choices is valuable.

6.5 Isolation: Shape Flexibility

The accelerators compared are as follows: InFlex-0001, using a square PE array, PartFlex-0001-A, a flexible PE array composed of a modular 16×16 PE-array building block, PartFlex-0001-B, a flexible PE array with a 4×4 building block, and FullFlex-0001.

Map Space. C_X is decided by the size of the PE array. A_X for the PartFlex-0001 is the number of different array shapes that can be composed by the smaller building block. A smaller building block can enable more fine-grained PE array shape exploration and thus exposes a larger configuration space, which can be observed by the different size of A_X for the two partially flexible accelerators in Fig. 11(b).

Performance. When the size of parallelism dimension is larger than the size of the building block (32×32 , 16×16 , 4×4 , or flexible), we need to fold the computation. When they are not perfectly divisible, the last fold of computation will introduce compute under-utilization, as shown in Layer-15 and Layer-16 of PartFlex-0001-A in Fig. 11(a). Interestingly, PartFlex-0001-B can mostly reach optimum performance, since most of the layers have K, C dimension sizes that are multiples of 4. Note that we use the default K-C parallelism strategy in this experiment. PartFlex-0001-B reaches almost the same performance as the fully-flexible one with only 6% flexibility. Overall, we observe that full flexibility achieve speedup of $1.05 \times$, $1.02 \times$, and $1.001 \times$ over the three baselines, respectively.

Sensitivity Analysis: Array Size. A larger number of PEs can allow more array configuration options and hence has larger hardware Flexion, as shown in Fig. 12. We can also observe the diminishing return around 45×45 to 64×64 points when the provided number of PEs is starting to become larger than the maximum K-C parallelism that some of the layers can leverage, leading to under utilization of the PEs.

Takeaways. We observe the flexibility in array shape is highly crucial for utilization across layers. However, it is highly sensitive to the dimensions being mapped spatially. This indicates that flexibility in P and S go hand-in-hand (more discussion on this in Section 7). One key observation is that a partially flexible accelerator using multiple fixed-size small arrays (e.g., 4x4) reaches almost the same performance as the fully-flexible one. This observation is consistent with some design choices in the existing accelerator designs, e.g., the 4 × 4 matrix multiplier in CUDA tensor cores [4]. The partially flexible accelerators are also sufficient for many manual-design CNNs, e.g., ResNet50 [20], VGG16 [42], whose dimension sizes are mostly multiples of 64 in K, C dimensions. However, shape flexibility could become increasingly important because of two use cases, as follows. (1) More and more neural network designs are relying on NAS, which could generate more diverse layer shapes. MnasNet is one such NAS network, but it is highly modularized, while more diverse

networks could be developed in future to target different applications. (2) Pruning techniques can change layer shapes in diverse ways.

7 EVALUATION II: ACCELERATOR FUTURE-PROOFING

In this experiment, we demonstrate how this work enables a first-of-its-kind experiment: assessing the advantage of adding flexibility to an accelerator at design time in terms of future-proofing, quantifying the trade-off between hardware cost and the performance gain on "future" workloads. To start, we design an inflexible accelerator tailored for Alexnet, InFlex-0000-Alexnet-Opt, by finding a fixed configuration of TOPS to optimize its runtime. This represents an accelerator such as Eyeriss [12] which was a optimized for a fixed workload in a certain year (~2014).

Fixed Alexnet-Opt Accel on Future Models. We now progress our accelerator into the future. When new neural network models (model X) are introduced. We design InFlex-0000-X-Opts, representing a new accelerator optimized just for this new workload—though for comparison purposes we always use roughly the same number of hardware resources as InFlex-0000-Alexnet-Opt. First, we use the InFlex-0000-Alexnet-Opt to run the future models and compare the performance witht the future model (X) optimized InFlex-0000-X-Opt. We observe that InFlex-0000-Alexnet-Opt can achieve similar performance to InFlex-0000-X-Opt when X is a CNN model, which follows the intuition since Alexnet is also a CNN, which might share similar characteristics. However, for fully-connected (FC) layer dominated models such as BERT, DLRM, and NCF, we see 1/0.25 = 4x to 1/0.07=14x slowdown. This shows that a fixed configuration can hurt performance when new models start to show notably different characteristics (e.g., CONV- vs FC-dominated).

Single-Axes Flexible Alexnet-Opt Accel on Future Models. Next, we go back to the "past" to explore the effect of adding single-axes flexibility when designing Alexnet-Opt accelerator, as shown in rows 3-6 of Fig. 13.

We see that in MnasNet, FullFlex-1000-Alexnet-Opt is $1.00/0.57 = 1.75 \times$ better than InFlex-0000-Alexnet-Opt. *We found tile flexibility to be crucial for the three CNNs and BERT*. However, it is not as effective in DLRM and NCF. The reason is as follows. In this experiment, InFlex-0000-Alexnet-Opt found Y-K parallelism to be optimal for Alexnet, and this stays valid for other CNNs, too. BERT is composed of matrix-matrix multiplications, the GEMM (M,N,K) dimensions got mapped to the (K_conv,C,Y) dimensions of the CONV accelerator respectively. The Y-K parallelism in the accelerator helped BERT as well. However, DLRM and NCF use matrix-vector multiplications, where the K dimension is 1 for the vector. Mapping them on the InFlex-0000-Alexnet-Opt accelerator leads to under-utilization along the Y-parallelism dimension of the accelerator, leading to loss in performance. In order to run DLRM and NCF efficiently, the ability to switch to other parallelism strategies become crucial.

Multiple-Axes Flexible Alexnet-Opt Accel on Future Models. There are 10 other variants of combinatorial flexible accelerators, as shown in row 7-16 of Fig. 13. We see that (P)+(S) together (0011) work well and show significant performance gain compared to the separated counterpart (0010, 0001). (T)+(P) with only two flexibility axes enabled can achieve comparable performance with FullFlex-1111-Alexnet-Opt in multiple cases. Interestingly, we see that (T)+(O) works much worse than (T)+(P). This shows that (T), which individually works well still needs to find the right other flexibility axes to have another level of performance boost.

Fully Flexible Alexnet-Opt Accel on Future Models. Finally, we investigate two kinds of flexible accelerators: FullFlex-1111-Alexnet-Opt, and PartFlex-1111-Alexnet-Opt. PartFlex-1111-Alexnet-Opt uses the configurations described in the evaluations in Section 6 (using variant A for Shape).

While more flexibility support introduces larger area cost, it is noteworthy that it brings with potential to leverage better runtime *even in the original AlexNet*. This is because AlexNet itself has a

lot of variation across its layers. *A fixed accelerator that optimizes for the average-case across all layers may miss the potential to reach the best case for each individual layer.* For e.g., FullFlex-1111-Alexnet-Opt speeds up the baseline InFlex-0000-Alexnet-Opt by 1/0.06=16.7x times in Alexnet. Furthermore, FullFlex-1111-Alexnet-Opt constantly achieves better performance than InFlex-0000-X-Opts.

Takeaways. Accelerators optimized for certain layers (such as CONV2D) are more future proof, even if the layer dimensions of future models change. This is because most models show abundant parallelism across the input and output channels. Often tile flexibility seems sufficient to capture most benefits. However, for FC-dominated layer types, flexibility in parallelism and shape starts reaping more benefits since there are fewer dimensions and GEMMs can be irregular. As models with several other layer types emerge (e.g., FC/DWCONV/Attention/...), investing in flexibility is valuable.

8 RELATED WORK

Flexible DNN accelerators. Flexibility support in DNN accelerators is an active topic of research, with several "flexible" accelerators proposed over the years [13, 19, 31, 32, 39]. The taxonomy from this work can tease out the specific axes of flexibility each provided and enable quantitative comparisons.

Mapping Tools. Several mapping tools have been proposed to search through map-spaces of flexible accelerators. Section 2.3 presents the state-of-the-art. In this work, we extend GAMMA [27] to support flexibility-aware optimization.

9 CONCLUSIONS AND FUTURE WORK

This work demonstrates that increased formalism of the notion of flexibility leads to concrete benefits. Namely: (A) the ability to taxonomize existing accelerators along flexibility axes, (B) the ability to precisely quantify the shape of a map-space against a theoretical upper-bound, and (C) the ability to integrate flexibility into existing DSE flows. We identify that in several cases, partial flexibility along some of the axes is sufficient for capturing an optimized mapping. Most significantly, our evaluation of a 2014-style accelerator demonstrates that a design-time investment in flexibility can lead to concrete improvements in future-proofing after deployment time.

Of course, the field of deep learning is changing incredibly rapidly. Turing Complete platforms such as GPUs and CPUs will undoubtedly continue to play a key and irreplaceable role as the main platforms that allow machine learning experts to rapidly innovate. However, we believe that it is equally important that the most successful neural network architectures can be accelerated via hardware specialization, without ASIC designers risking complete obsolescence after deployment. Ultimately, the key reason flexible mapping support can be omitted from hardware designs is not area or energy, but simply design and verification effort, as well as mapper toolchain support. We hope that this work will ultimately lead to more research that allows architects to quantify the benefits of "future-proofing" with the same level of precision as present-day budgets. Finally, while most existing DSE framework focus on finding the "optimal design point", this work, armed with the ability to systematically constructing map-spaces of fully/partially flexible accelerators, opens an interesting future avenue of research to identify "regions of optimal (i.e., high-performing) design space", leading to better explainability of the DSE algorithms and interpretability of the accelerator deign space.

We believe this work will help both compiler developers and architects systematically develop flexibility-aware accelerator designs. Given that flexibility is often not added in HW to simplify engineering effort, we envision partial flexibility being ripe for future work. In future, we hope our analysis can be expanded to cover even more complex aspects of data orchestration and generalized beyond DNNs.

10 ACKNOWLEDGMENT

This work was supported by NSF Award 1909900.

REFERENCES

- [1] https://github.com/maestro-project/gamma.
- $\cite{21} https://https://github.com/georgia-tech-synergy-lab/maeri.$
- [3] Freepdk15:contents. https://www.eda.ncsu.edu/wiki/FreePDK15:Contents, 2019.
- [4] Cuda c++ programming guide. 2021.
- [5] Manoj Alwani, Han Chen, Michael Ferdman, and Peter Milder. Fused-layer cnn accelerators. In 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), pages 1–12. IEEE, 2016.
- [6] Eunjin Baek, Dongup Kwon, and Jangwoo Kim. A multi-neural network acceleration architecture. In 2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA), pages 940–953. IEEE, 2020.
- [7] James Bergstra, Daniel Yamins, and David Daniel Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. 2013.
- [8] Prasanth Chatarasi, Hyoukjun Kwon, Natesh Raina, Saurabh Malik, Vaisakh Haridas, Angshuman Parashar, Michael Pellauer, Tushar Krishna, and Vivek Sarkar. Marvel: A data-centric compiler for dnn operators on spatial accelerators. arXiv preprint arXiv:2002.07752, 2020.
- [9] Mark Chen, Alec Radford, Rewon Child, Jeffrey Wu, Heewoo Jun, David Luan, and Ilya Sutskever. Generative pretraining from pixels. In International Conference on Machine Learning, pages 1691–1703. PMLR, 2020.
- [10] Tianqi Chen, Lianmin Zheng, Eddie Yan, Ziheng Jiang, Thierry Moreau, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. Learning to optimize tensor programs. In Advances in Neural Information Processing Systems, pages 3389–3400, 2018.
- [11] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. ACM SIGARCH Computer Architecture News, 44(3):367–379, 2016.
- [12] Yu-Hsin Chen, Tushar Krishna, Joel S Emer, and Vivienne Sze. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal of Solid-State Circuits*, 52(1):127–138, 2016.
- [13] Yu-Hsin Chen, Tien-Ju Yang, Joel Emer, and Vivienne Sze. Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 9(2):292–308, 2019.
- [14] Shail Dave, Youngbin Kim, Sasikanth Avancha, Kyoungwoo Lee, and Aviral Shrivastava. Dmazerunner: Executing perfectly nested loops on dataflow accelerators. ACM Transactions on Embedded Computing Systems (TECS), 18(5s):1–27, 2019.
- [15] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805, 2018.
- [16] Zidong Du, Robert Fasthuber, Tianshi Chen, Paolo Ienne, Ling Li, Tao Luo, Xiaobing Feng, Yunji Chen, and Olivier Temam. Shidiannao: Shifting vision processing closer to the sensor. In Proceedings of the 42nd Annual International Symposium on Computer Architecture, pages 92–104, 2015.
- [17] William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. arXiv preprint arXiv:2101.03961, 2021.
- [18] Mingyu Gao, Xuan Yang, Jing Pu, Mark Horowitz, and Christos Kozyrakis. Tangram: Optimized coarse-grained dataflow for scalable nn accelerators. In Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, pages 807–820, 2019.
- [19] Soroush Ghodrati, Byung Hoon Ahn, Joon Kyung Kim, Sean Kinzer, Brahmendra Reddy Yatham, Navateja Alla, Hardik Sharma, Mohammad Alian, Eiman Ebrahimi, Nam Sung Kim, et al. Planaria: Dynamic architecture fission for spatial multi-tenant acceleration of deep neural networks. In 2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), pages 681–697. IEEE, 2020.
- [20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 770–778, 2016.
- [21] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In Proceedings of the 26th international conference on world wide web, pages 173–182, 2017.
- [22] Kartik Hegde, Po-An Tsai, Sitao Huang, Vikas Chandra, Angshuman Parashar, and Christopher W Fletcher. Mind mappings: Enabling efficient algorithm-accelerator mapping space search extended abstract.
- [23] Kartik Hegde, Jiyong Yu, Rohit Agrawal, Mengjia Yan, Michael Pellauer, and Christopher W. Fletcher. Ucnn: Exploiting computational reuse in deep neural networks via weight repetition. In *Proceedings of the 45th Annual International Symposium on Computer Architecture*, ISCA '18, page 674–687, 2018.
- [24] Qijing Huang, Minwoo Kang, Grace Dinh, Thomas Norell, Aravind Kalaiah, James Demmel, John Wawrzynek, and Yakun Sophia Shao. Cosa: Scheduling by constrained optimization for spatial accelerators. arXiv preprint

arXiv:2105.01898, 2021.

- [25] Norman P Jouppi, Doe Hyun Yoon, George Kurian, Sheng Li, Nishant Patil, James Laudon, Cliff Young, and David Patterson. A domain-specific supercomputer for training deep neural networks. *Communications of the ACM*, 63(7):67–78, 2020.
- [26] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings* of the 44th Annual International Symposium on Computer Architecture, pages 1–12, 2017.
- [27] Sheng-Chun Kao and Tushar Krishna. Gamma: Automating the hw mapping of dnn models on accelerators via genetic algorithm. In *ICCAD*, 2020.
- [28] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of StyleGAN. In Proc. CVPR, 2020.
- [29] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems, 25:1097–1105, 2012.
- [30] Hyoukjun Kwon, Prasanth Chatarasi, Vivek Sarkar, Tushar Krishna, Michael Pellauer, and Angshuman Parashar. Maestro: A data-centric approach to understand reuse, performance, and hardware cost of dnn mappings. *IEEE Micro*, 40(3):20–29, 2020.
- [31] Hyoukjun Kwon, Ananda Samajdar, and Tushar Krishna. Maeri: Enabling flexible dataflow mapping over dnn accelerators via reconfigurable interconnects. ACM SIGPLAN Notices, 53(2):461–475, 2018.
- [32] Wenyan Lu, Guihai Yan, Jiajun Li, Shijun Gong, Yinhe Han, and Xiaowei Li. Flexflow: A flexible dataflow accelerator architecture for convolutional neural networks. In 2017 IEEE International Symposium on High Performance Computer Architecture (HPCA), pages 553–564. IEEE, 2017.
- [33] Maxim Naumov, Dheevatsa Mudigere, Hao-Jun Michael Shi, Jianyu Huang, Narayanan Sundaraman, Jongsoo Park, Xiaodong Wang, Udit Gupta, Carole-Jean Wu, Alisson G. Azzolini, Dmytro Dzhulgakov, Andrey Mallevich, Ilia Cherniavskii, Yinghai Lu, Raghuraman Krishnamoorthi, Ansha Yu, Volodymyr Kondratenko, Stephanie Pereira, Xianjie Chen, Wenlin Chen, Vijay Rao, Bill Jia, Liang Xiong, and Misha Smelyanskiy. Deep learning recommendation model for personalization and recommendation systems. *CoRR*, abs/1906.00091, 2019.
- [34] Maxim Naumov, Dheevatsa Mudigere, Hao-Jun Michael Shi, Jianyu Huang, Narayanan Sundaraman, Jongsoo Park, Xiaodong Wang, Udit Gupta, Carole-Jean Wu, Alisson G Azzolini, et al. Deep learning recommendation model for personalization and recommendation systems. arXiv preprint arXiv:1906.00091, 2019.
- [35] NVIDIA. Nvdla deep learning accelerator. http://nvdla.org, 2017.
- [36] Angshuman Parashar, Priyanka Raina, Yakun Sophia Shao, Yu-Hsin Chen, Victor A Ying, Anurag Mukkara, Rangharajan Venkatesan, Brucek Khailany, Stephen W Keckler, and Joel Emer. Timeloop: A systematic approach to dnn accelerator evaluation. In 2019 IEEE international symposium on performance analysis of systems and software (ISPASS), pages 304–315. IEEE, 2019.
- [37] Michael Pellauer, Yakun Sophia Shao, Jason Clemons, Neal Crago, Kartik Hegde, Rangharajan Venkatesan, Stephen W Keckler, Christopher W Fletcher, and Joel Emer. Buffets: An efficient and composable storage idiom for explicit decoupled data orchestration. In Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, pages 137–151, 2019.
- [38] Mateja Putic, Swagath Venkataramani, Schuyler Eldridge, Alper Buyuktosunoglu, Pradip Bose, and Mircea Stan. Dyhard-dnn: Even more dnn acceleration with dynamic hardware reconfiguration. In *Proceedings of the 55th Annual Design Automation Conference*, pages 1–6, 2018.
- [39] Eric Qin, Ananda Samajdar, Hyoukjun Kwon, Vineet Nadella, Sudarshan Srinivasan, Dipankar Das, Bharat Kaul, and Tushar Krishna. Sigma: A sparse and irregular gemm accelerator with flexible interconnects for dnn training. In 2020 IEEE International Symposium on High Performance Computer Architecture (HPCA), pages 58–70. IEEE, 2020.
- [40] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 4510–4520, 2018.
- [41] Yakun Sophia Shao, Jason Clemons, Rangharajan Venkatesan, Brian Zimmer, Matthew Fojtik, Nan Jiang, Ben Keller, Alicia Klinefelter, Nathaniel Pinckney, Priyanka Raina, et al. Simba: Scaling deep-learning inference with multi-chipmodule-based architecture. In Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture, pages 14–27, 2019.
- [42] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556, 2014.
- [43] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 2820–2828, 2019.
- [44] Shmuel Winograd. Arithmetic Complexity of Computations, chapter 7, pages 71-92.

Proc. ACM Meas. Anal. Comput. Syst., Vol. 6, No. 2, Article 41. Publication date: June 2022.

41:22

- [45] Xuan Yang, Mingyu Gao, Qiaoyi Liu, Jeff Setter, Jing Pu, Ankita Nayak, Steven Bell, Kaidi Cao, Heonjae Ha, Priyanka Raina, et al. Interstellar: Using halide's scheduling language to analyze dnn accelerators. In Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems, pages 369–383, 2020.
- [46] Yaqi Zhang, Nathan Zhang, Tian Zhao, Matt Vilim, Muhammad Shahbaz, and Kunle Olukotun. Sara: Scaling a reconfigurable dataflow accelerator. In 2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA), pages 1041–1054. IEEE, 2021.

Received February 2022; revised March 2022; accepted April 2022