Flexion: A Quantitative Metric for Flexibility in DNN Accelerators

Hyoukjun Kwon*, Michael Pellauer†, Angshuman Parashar†, Tushar Krishna*
* Georgia Institute of Technology † NVIDIA
* *hyoukjun@gatech.edu, tushar@ece.gatech.edu* † *{mpellauer, aparashar} @nvidia.com*

Abstract—Dataflow and tile size choices, which we collectively refer to as mappings, dictate the efficiency (i.e., latency and energy) of DNN accelerators. Rapidly evolving DNN models is one of the major challenges for DNN accelerators since the optimal mapping heavily depends on the layer shape and size. To maintain high efficiency across multiple DNN models, flexible accelerators that can support multiple mappings have emerged. However, we currently lack a metric to evaluate accelerator flexibility and quantitatively compare their capability to run different mappings. In this work, we formally define the concept of flexibility in DNN accelerators and propose *flexion* (flexibility fraction), flexion, which is a quantitative metric of mapping flexibility on DNN accelerators. We codify the formalism we construct and evaluate the flexibility of accelerators based on Eyeriss, NVDLA, and TPUv1. We show that Eyeriss-like accelerator is 2.2× and 17.0× more flexible (i.e., capable of running more mappings) than NVDLA and TPUv1-based accelerators on selected ResNet-50 and MobileNetV2 layers. This work is the first work to enable such a quantitative comparison of the flexibility of accelerators.

1 Introduction

Domain-specific accelerators exhibit significantly increased efficiency for their target workload but are not Turing Complete. However, many non-programmable accelerators do retain a degree of configurability, especially related to *data orchestration*: namely, the choice of scheduling data movement, tile size, buffer allocation, and computation ordering. We term the degree of data orchestration configurability as *accelerator flexibility* and distinguish it from the degree of reconfigurability based on functionality in reconfigurable logic (i.e., FPGAs). Flexibility is particularly important for deep-learning accelerators, as different neural-network layers can vary significantly in size and shape. Relying on a single problem dimension (e.g., rows of weights, output channels, etc.) for parallelism or data reuse can result in severe under-utilization when the dimension is too small. For example, in an early layer in ResNet-50 (CONV2_1) has 56x56 activation with 64 channels, but in a late layer in ResNet-50 (CONV5_3) has 7x7 activation with 2048 channels. A mapping that exploits parallelism on activation can result in severe underutilization on the late layer, and vice versa for one that exploits parallelism on channels on the early layer. There is extensive evidence that the optimal data orchestration configuration can vary both across and within networks [5], [8].

Furthermore, the field of neural networks is changing rapidly, with new networks being proposed at a breakneck pace. Architects cannot see the future, and so must make decisions today based on finite benchmark sets and a limited ability to predict trends in neural networks, knowing that it may be several years before their design reaches the field. From this point of view, flexibility is desirable for "future-proofing" as well, but whether or not this investment pays off can only be quantified post-facto.

We thank Sheng-Chun (Felix) Kao for help with performance results in Table 1.

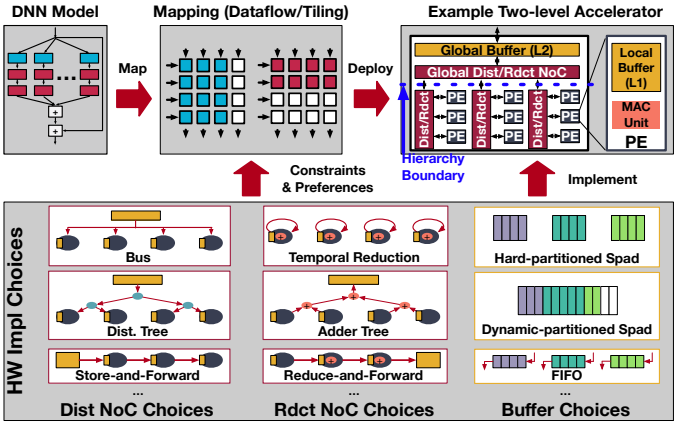

Fig. 1. An overview of the impact of hardware implementation choices on DNN deployment on an accelerator.

This leads to an intractable computer architecture problem that we term the Accelerator Mapping Flexibility Conundrum. Namely, investing in hardware features for flexibility consumes resources that could be applied elsewhere–area, energy, and significant engineering effort in design/verification and configuration toolchain. Therefore, even if a strong predictive case can be made for flexibility features, they often end up "on the cutting room floor" when faced with real-world budgeting of area/energy/effort.

In order to break this conundrum, this paper proposes a first-of-its-kind quantitative metric of accelerator flexibility. In contrast with Turing Completeness, our goal is not binary yes/no judgement, but rather a *flexibility fraction* or *flexion*[1] of the general form:

$$\frac{Mappings_{Achievable}}{Mappings_{Possible}}$$

Where: **(1)** $Mappings_{Possible}$ derives only from the network layer being mapped, and is the same across all accelerators (with some key restrictions to avoid infinity that we detail), and **(2)** $Mappings_{Achievable}$ is the number of those mappings that the specific accelerator design being measured can exploit.

Therefore, within the limit, a fully-fixed accelerator will approach 0, and a fully-flexible accelerator will approach 1. The flexion formulation enables architecture-time comparison across potential design points of the same accelerator, and also comparison across accelerators. Ultimately, our metric can be paired with

1. Pronounced "fleck-shun." In biology this term refers to the act of bending a joint or limb, so we find it particularly suitable for a flexibility metric.

TABLE 1
An example trade-off study of 256-PE-Eyeriss-, NVDLA-, and TPU-like accelerators (EL, NL, and TL) implemented in RTL and synthesized using 28nm library. We run MAESTRO [5] with optimized mapping generated by GAMMA [10] to estimate the latency. We quantify the flexibility of each accelerator Flexion. RC layers are from Resnet50 [2], and MB layers are from MobileNetV2 [9] as listed in Section 4.

Acc.	Area(mm^2)	Power(mW)	Layer	Latency	Flexion
EL	5.13	188.33	RC2_3	4.92E+03	2.23E-04
			RC3_2	2.02E+04	7.53E-05
			RC5_2	3.27E+04	1.57E-05
			MB1_DW	4.17E+03	4.15E-06
			MB5_DW	1.14E+03	4.57E-06
			MB6_R	2.34E+02	2.33E-03
			Avg.	1.06E+04	4.77E-05
NL	3.85	100.18	RC2_3	1.39E+05	6.24E-07
			RC3_2	3.96E+04	5.29E-06
			RC5_2	8.43E+03	1.10E-05
			MB1_DW	5.11E+03	7.69E-08
			MB5_DW	4.02E+03	2.90E-06
			MB6_R	1.26E+03	7.71E-03
			Avg.	3.29E+04	6.30E-06
TL	1.82	40.79	RC2_3	5.67E+04	1.73E-08
			RC3_2	6.79E+04	1.49E-07
			RC5_2	1.25E+04	3.11E-07
			MB1_DW	5.08E+03	7.69E-08
			MB5_DW	1.94E+03	2.90E-06
			MB6_R	3.29E+02	7.71E-03
			Avg.	2.41E+04	1.05E-06

ell-understood area/energy quantification techniques to create a comprehensive flexibility cost/benefit analysis of buffering, control, and network-on-chip hardware features, as an example in Table 1.

2 Background

Each layer of a DNN can be expressed as a multi-dimensional loop nest over the input and weight tensors. When running a layer on a DNN accelerator, we collectively refer to the execution order of computations, parallelized dimensions, and the number of tiling levels as the accelerator's *dataflow*. We refer to the dataflow with specific tile sizes at each level as *mapping*.

Since all DNN accelerators compute DNNs, the definition of flexibility based on functionality (i.e., what they can compute) does not provide distinction among flexible accelerators. Instead of supporting more applications, the focus of flexible accelerators is providing capabilities for supporting various mapping styles [3], [4]. Therefore, we also focus on the number of supported mappings to compute a target DNN. Considering that the total number of possible mappings on unconstrained hardware depends on layer operation and sizes, we define the flexibility at a layer granularity.

We identify the following four components of mapping flexibility, abbreviated as TOPS (i.e., as a synonym of the performance metric, TOPS): **(1) Loop Tiling (T):** The number of tiling levels and tile sizing. **(2) Loop Order (O):** The order of data dimension iterations. **(3) Loop Parallelization (P):** The data dimension to be parallelized. Represents the *spatial* partitioning (i.e., partitioning data over processing elements (PEs)) of data.) **(4) Array Shape (S):** The shape of the accelerator array. 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.

Based on the four components, we can define flexibility on each component, and define a mapping flexibility, we term as **Flexion** as the cross-product of all of the four components, which represents the flexibility fraction we discussed in Section 1. We discuss the definition and computation of flexibility fraction next.

3 Flexibility Fraction, or Flexion

Hardware implementation choices impose constraints on available mappings and present preference to a specific set of mappings, as illustrated in Figure 1. Based on the observation, we define flexion based on the number of available mappings with hardware constraints over the total number of available mappings for given hierarchy levels without hardware constraints. To formalize the available and achievable mapping counts, we model accelerators to be a hierarchy of temporal/spatial distribution (i.e., buffer and distribution NoCs) and reduction elements (i.e., reduction NoCs). Figure 1 shows an example accelerator of two-level hierarchy where the level boundary is highlighted by a blue dotted line. We model layer size and tile size to be a list of layer dimension-size tuples. Based on the base definitions, we discuss how hardware choices impact mapping choices next.

3.1 Hardware Choice and Constraints on Mapping

We focus on hard constraints on mappings to identify the number of legal mappings under hardware constraints (i.e., flexibility), not finding the best mapping, which is an open research question [10]. Figure 2 summarizes such hard constraints on mapping based on non-circular FIFO buffers and temporal reduction.

FIFO's constraints are based on convolutional reuse across sliding windows, which requires breaking the FIFO behavior to read data accessed in the past. Temporal reduction refers to a reduction network-on-chip (NoC) supporting reduction only over time, not across PEs [5], where its constraints are based.

Based on the constraints we discussed and buffer sizes, we can determine if a mapping is valid on an accelerator (e.g., check tile sizes if data tiles fit into memories). We codify the mapping validity checker and use them in our flexibility evaluation framework used for case studies in Section 4. We can use such a checker to count the number of valid mappings on a given accelerator, on which the mapping flexibility we define is based.

Next, we discuss how to compute the total number of choices for each flexibility component (TOPS) without constraint and introduce the definition of our flexibility metric.

3.2 The Number of Mappings without Constraints

Based on the four components of flexibility, TOPS (**T**ile size, loop **O**rder, **P**arallel dimension, and **S**hape) we discussed in Section 2, we first define total number of choices for each component without hardware constraints for a given number of hierarchies in an accelerator.

Definition 1. **Tiling Choices at Hierarchy Level H_{lv}**
At cluster level H_{lv}, given the tile size of $\mathrm{T}(H_{lv})$, the number of tile size choices for level $H_{lv}-1$, $\tau(\mathrm{T}(H_{lv}), H_{lv}-1)$, is as follows:

$$\tau(\mathrm{T}(H_{lv}), H_{lv}-1) = \prod_{\forall(d,sz)\in \mathrm{T}(H_{lv})} sz$$

Note that the number of tiling choices depends on the tile size at the upper-level (i.e., tile size at lower level $\leq$ tile size at the upper level) Therefore, we first define the relationship between two adjacent tile sizes in Definition 1. Based on Definition 1, we define the entire number of tiling choices in Definition 2.

Definition 2. **Tiling Choices**
The number of tiling choices for a given layer size L_{sz} and number of on-chip cluster levels N_H, the number of

Hardware		Implication to Mapping								Costs and Benefits		
Hardware Type	Hardware Choice	Tile Size						Spatial Dimension	Iteration Order	Performance Benefits	Performance Disadvantages	Hardware Cost
		K	C	Y	X	R	S					
Buffer	FIFO	-	-	≤Sz(R)	≤Sz(S)	-	-	-	-	O(1)	-	Control: O(1)
	Scratchpad	-	-	-	-	-	-	-	-	O(1)	-	Control: O(n)
Distribution NoC	Bus	-	-	-	-	-	-	C Unpreferred	SpDim: inner-most position unpreferred	O(T)/m*		O(n)
	Crossbar	-	-	-	-	-	-	-	-	O(1)	-	$O(n^2)$
	Mesh	-	-	-	-	-	-	-	-	-	- Link Conflict - Hop-by-hop traversal	O(n)
	Tree	-	-	-	-	-	-	C Unpreferred	SpDim: inner-most position unpreferred	Multicast Support	-	O(n log(n))
	Store-and-Forward	-	-	-	-	-	-	-	-	Multicast Support	High Latency (O(n))	O(n)
Reduction NoC	Temporal Reduction	-	-	-	-	-	-	C/R/S Prohibited	-	-	Low Reduction Bandwidth	O(1)
	Adder Tree	-	-	-	-	-	-	C/R/S Preferred	-	High BW Low Latency	-	O(n log(n))
	Reduce-and-Forward	-	-	-	-	-	-	C/R/S Preferred	-	Pipelined Reduction	High Latency	O(n)

Fig. 2. A summary of hardware choices and their implication to mapping. In the implication to mapping columns, red texts describe the hard constraints, and plain texts describe the preference for better efficiency. K/C refer to output/input channels, Y/X refer to input row/column, and R/S refer to filter row/column.

tiling choices, $\tau_{all}(L_{sz}, N_H)$, is as follows:

$$\tau_{all}(L_{sz}, N_H) = \tau_{rec}(T_{sz} = L_{sz}, H_{lv} = N_H) = \sum_{\forall \mathrm{T}(H_{lv})} \tau_{rec}(\mathrm{T}(H_{lv}), H_{lv} - 1)$$

where $\tau_{rec}(T_{sz}, 0) = 1$

Definition 1 indicates that the number of available tile size at a lower level cluster is constrained by the tile size at the upper level cluster. For example, if a mapping assigns indices of a dimension α in the range of $[0, \mathrm{T}(H_{lv})[\alpha])$, one level below cluster cannot have tile size larger than $\mathrm{T}(H_{lv})[\alpha]$ (i.e., $\mathrm{T}(H_{lv}-1)[\alpha] \leq \mathrm{T}(H_{lv})[\alpha]$) because of out-of-range indices. Therefore, the number of choices depends on the tile sizes on the upper-level cluster, which results in a recursive definition as shown in Definition 2. Also, note that the definition covers entire tile sizes including those do not divide the upper level tile sizes. For example, if $\mathrm{T}(3)[\alpha] = 64$, $\mathrm{T}(2)[\alpha]$ can be any number between 1 and 64.

Definition 3. **Loop Order Choices**

Given a layer size, L_{sz}, and the number of on-chip cluster levels, N_H, then loop order choices, $\omega(L_{sz}, N_H)$, is as follows:

$$\omega(L_{sz}, N_H) = (len(L_{sz})!)^{N_H}$$

In Definition 3, we consider permutation at each hierarchy level for all the independent data dimensions. For example, in CONV2D without batches, six dimensions lead to 6! choices at each level. The iteration order choice at each level is independent, so we compute the power of choices at each level.

Definition 4. **Parallel Dimension Choices**

Given a layer size, L_{sz}, and the number of on-chip hierarchy levels (N_H) the number of parallel dimension choices (π) is defined as follows:

$$\pi(L_{sz}, N_H) = \prod_{0 \leq H_{lv} < N_H} \binom{Len(L_{Sz}) - H_{lv}}{1}$$

In this definition, we exclude redundant cases that spatially partition a data dimension multiple times at multiple hierarchy levels since it is equivalent to a flattened mapping. For example, if we try to parallelize output channels across PE rows and columns (twice) on a 4x4 PE array, the resulting mapping is equivalent to parallelizing output channel across a 1D PE array with 4x4=16 PEs. They result in different performance and cost, but they are equivalent from the logical mapping's perspective. We do not count them separately when we estimate the flexibility.

Definition 5. **Shape choices for a 2D PE array**

Given an accelerator with N_{PE} of PEs in a flexible shape 2D PE array, the number of shape choices $\sigma(Acc)$ is defined as follows:

$$\sigma(Acc) = |\{r | r = \lfloor \frac{N_{PE}}{i} \rfloor, 1 \leq i \leq N_{PE}, i \in \mathbb{N}\}|$$

In Definition 5, we count all the possible combination of the aspect ratio of a fully-flexible shape 2D PE array, which can reconfigure the logical aspect ratio of the array. We separately count symmetric pairs (e.g., 2×4 and 4×2) since some architectures have specific features coupled with a specific array dimension (e.g., adder trees only support row-wise reduction).

Now we define component-wise flexibility based on the number of possible mappings with and without constraints.

3.3 Flexibility Metric

Definition 6. **Component-wise Flexibility**

$$Flex_{tiling} = \frac{\tau_{all-supported}(Acc, L_{sz}, N_H)}{\tau_{all}(L_{sz}, N_H)}$$

$$Flex_{order} = \frac{\omega_{supported}(Acc, L_{sz}, N_H)}{\omega(L_{sz}, N_H)}$$

$$Flex_{spatial} = \frac{\pi_{supported}(Acc, L_{sz}, N_H)}{\pi(L_{sz}, N_H)}$$

$$Flex_{shape} = \frac{\sigma_{supported}(Acc)}{\sigma_{all}(Acc)}$$

where $\tau_{all-supported}$, $\omega_{supported}$, $\pi_{supported}$, and $\sigma_{supported}$ refer to the number of available tile sizes, iteration orders, spatial (or, parallel) dimensions, and array shape choices by an accelerator Acc.

$Flex_{tiling}$, $Flex_{order}$, $Flex_{parallel}$, and $Flex_{shape}$ refer to tiling, iteration order, parallel dimension, and array shape flexibility, respectively.

In component-wise flexibility definitions listed in Definition 6, the denominator of each refers to algorithmic choices, which are determined by the layer size and the high-level architecture template of a target accelerator (the number of on-chip cluster levels; N_H). A high-level architecture template refers to an accelerator's organization, which includes the number of memory hierarchy levels and PE dimensionality without exact design parameters for them (e.g., memory size, number of PE rows/columns, etc.). The numerators ($\tau_{all-supported}$, $\omega_{supported}$, $\pi_{supported}$, and $\sigma_{supported}$) of

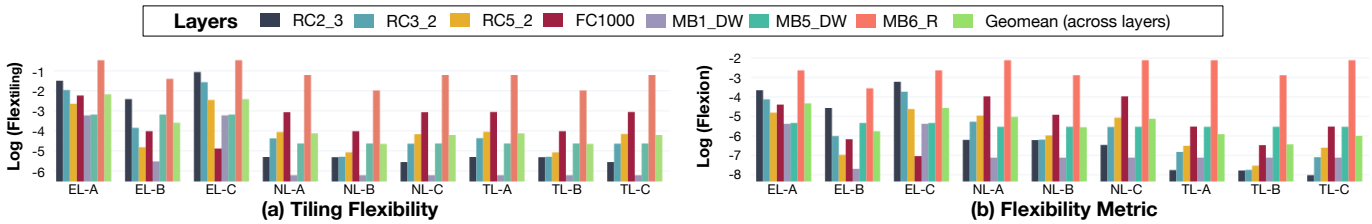

Fig. 3. Evaluated flexibility of accelerators listed in Table 2 on selected layers in Resnet50 [2]. EL, NL, and TL refer to Eyeriss, NVDLA, and TPUv1-like designs. Post-fix A, B, and C refer to three buffer size choices we discuss in Section 4. RC refers to CONV layers in Resnet50, and MB refers to the bottleneck in MobileNetv2. FC, DW, and R refers to fully-connected, depth-wise, and residual layers.

TABLE 2
A list of evaluated accelerators based on Eyeriss [1], NVDLA [6], and TPUv1 [7]. SF, RF, AT, and Spad refer to store-and-forward, reduce-and-forward, adder tree, and scratchpad.

Accelerator	D. NoC (L2/L1)	R. NoC (L2/L1)	Buffer (L2/L1)
Eyeriss-like	Bus / Bus	RF / Temporal	Spad / Spad
NVDLA-like	Bus / Bus	Temporal / AT	Spad / FIFO
TPUv1-like	SF / SF	Temporal / RF	Spad / FIFO

each refers to actually available choices constrained by a concrete accelerator (Acc) with all the design parameters.

Based on the component-wise flexibility definition, we define the mapping flexibility of an accelerator as the cross-product of all the component-wise flexibility.

Definition 7. **Mapping Flexibility (Flexion)**
Mapping flexibility is the product of all the component-wise flexibility (each ranges between 0 and 1):

$$Flex_{mapping} = Flex_{tiling} \times Flex_{order} \times Flex_{parallel} \times Flex_{shape}$$

Since we have defined ω, π, and τ_{all}, in this subsection, we need $\omega_{supported}, \pi_{supported}$, and $\tau_{all-supported}$ for computing the mapping flexibility. Therefore, we implement a mapping space enumerator based on the mapping validity checker we discussed in Subsection 3.1, which is based on a branch-and-bound mapping space iterator. We codify the computation of the flexibility metric we define in this paper and perform case studies.

4 Case Studies

We implement a framework that computes the number of possible mappings based on hardware choices and the total number of mappings without those constraints. Using the framework, we perform a case study on flexibility on selected early, late, and fully-connected layers in ResNet-50 [2] and two depth-wise convolution layers and residual in MobileNetV2 [9] on three accelerator styles we summarize in Table 2. For each accelerator style, we explore three variants of each accelerator style varying global buffer (L2) and local buffer in each PE (L1) sizes: A-150KB/1.5KB, B-150KB/150B, and C-60KB/1.5KB.

Figure 3 (a) and (b) show the evaluated tiling flexibility and overall flexibility metric. Spatial and loop order flexibilities are independent of buffer sizes. Eyeriss-like, NVDLA-like, and TPU-like designs scored for the $Flex_{spatial}$ of 0.25, 0.125, and 0.125, respectively, based on their reduction NoC and buffer choices. For loop orders, all accelerators are based on fixed loop orders, resulting in the $Flex_{order}$ of 0.027. Compiling all, Eyeriss-like accelerator is 2.2× and 17.0 × more flexible than NVDLA-like and TPU-like accelerators, on average across evaluated layers.
which led to significant differences in tiling flexibility between
Impact of Buffer Choice. Because we applied the same buffer size for each accelerator variants, such differences are based on the choice of distribution/reduction NoC and buffers. In particular, non-circulate FIFO imposes a strict restriction on available tile sizes, Eyeriss-like and other designs. Based on the flexible scratchpad in both L1 and L2 buffers, Eyeriss-like designs have 39.7× and 39.6× higher $Flex_{tilling}$, on average across evaluated layers.
Impact of Buffer Size. The variants B and C have smaller L1 and L2 buffer sizes than the variant A for each accelerator style. Normalizing the buffer size differences between L1 and L2, we observe that flexibility is 1.77× more sensitive on L2 buffer sizes.

We list up Flexion scores with latency and area/power information in Table 1.

5 Conclusion

In this work, we formally defined a first-of-its-kind quantitative metric of flexibility in DNN accelerators, flexion, which ranges from 0 for fully-fixed to 1 for fully-flexible accelerators. Flexion enables us to quantitatively evaluate the mapping flexibility of DNN accelerators and compare their capability of running mappings. Using Flexion, we showed that an Eyeriss-like accelerator we evaluate is 2.2× and 17.0 × more flexible (i.e., capable of running more mappings) than NVDLA and TPUv1-based accelerators on selected Resnet50 and MobileNetV2 layers. The case study showed that scratchpad provides dramatically higher tiling flexibility than FIFOs, and global buffer size is more critical for flexibility than local buffer sizes, which can guide the DNN accelerator design targeting future DNN models.

Flexion can be another pillar of DNN accelerator optimization as the amount of adaptivity to various layers in future DNN models combined with area/energy quantification techniques Such an approach enables a comprehensive flexibility-hardware cost-latency/energy benefit analysis of buffering, control, and network-on-chip hardware features.

Also, Flexion can be extended to layer fusion and sparsity, since the fundamental idea (i.e., $Mapping_{Achievable}$ / $Mapping_{possible}$) is generic. We believe this work will enable such works in the future.

References

[1] Y. Chen *et al.* Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. In *ISCA*, 2016.
[2] K. He *et al.* Deep residual learning for image recognition. In *CVPR*, 2016.
[3] K. Hegde *et al.* Morph: Flexible acceleration for 3d cnn-based video understanding. In *MICRO*, 2018.
[4] H. Kwon *et al.* Maeri: Enabling flexible dataflow mapping over dnn accelerators via reconfigurable interconnects. In *ASPLOS*, 2018.
[5] H. Kwon *et al.* Maestro: A data-centric approach to understand reuse, performance, and hardware cost of dnn mappings. *IEEE Micro*, 2020.
[6] NVIDIA. Nvdla deep learning accelerator. http://nvdla.org, 2017.
[7] N. P Jouppi *et al.* In-datacenter performance analysis of a tensor processing unit. In *ISCA*, 2017.
[8] A. Parashar *et al.* Timeloop: A systematic approach to dnn accelerator evaluation. In *ISPASS*, 2019.
[9] M. Sandler *et al.* Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*, 2018.
[10] K. Sheng-Chun *et al.* Gamma: Automating the hw mapping of dnn models on accelerators via genetic algorithm. In *ICCAD*, 2020.