Theme Article: Top Picks

MAESTRO: A Data-Centric Approach to Understand Reuse, Performance, and Hardware Cost of DNN Mappings

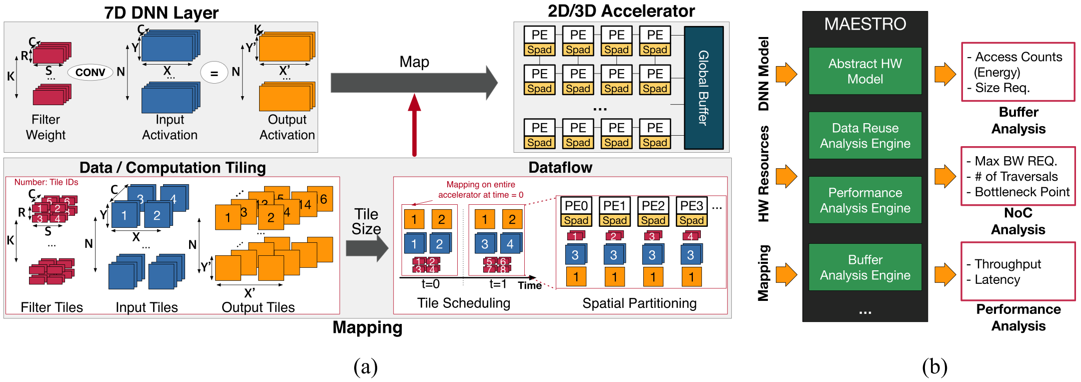
Hyoukjun Kwon, Prasanth Chatarasi, Vivek Sarkar, and Tushar Krishna
Georgia Tech

Michael Pellauer and Angshuman Parashar
NVIDIA Corp

***Abstract*—The efficiency of an accelerator depends on three factors—mapping, deep neural network (DNN) layers, and hardware—constructing extremely complicated design space of DNN accelerators. To demystify such complicated design space and guide the DNN accelerator design for better efficiency, we propose an analytical cost model, MAESTRO. MAESTRO receives DNN model description and hardware resources information as a list, and mapping described in a data-centric representation we propose as inputs. The data-centric representation consists of three directives that enable concise description of mappings in a compiler-friendly form. MAESTRO analyzes various forms of data reuse in an accelerator based on inputs quickly and generates more than 20 statistics including total latency, energy, throughput, etc., as outputs. MAESTRO's fast analysis enables various optimization tools for DNN accelerators such as hardware design exploration tool we present as an example.**

Digital Object Identifier 10.1109/MM.2020.2985963
Date of publication 22 April 2020; date of current version 22 May 2020.

0272-1732 © 2020 IEEE



(a)

(b)

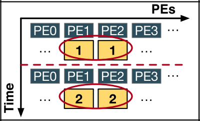
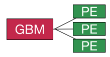
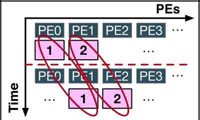
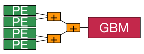
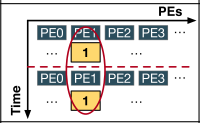
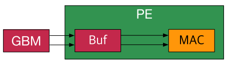
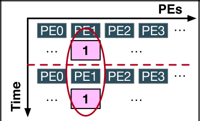
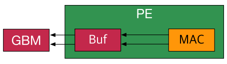
Figure 1. High-level overview of mapping a high-dimensional DNN layer (CONV2D in this figure) to an accelerator with 2-D PE array. Note that tile scheduling also needs to be done within spatial partitioning; we omit it for simplicity. (a) An Overview of Mapping CONV2D to an Accelerator. (b) High-level Tool flow of MAESTRO.

■ **Deep neural network** (DNN) inference accelerators achieve high performance by exploiting parallelism over hundreds of processing elements (PEs) and high energy efficiency by maximizing data reuse within PEs and on-chip scratchpads.[1–4] The efficiency (performance and energy efficiency) of a DNN accelerator depends on three factors depicted in Figure 1: 1) the workload (DNN layers), 2) the amount and type of available hardware resources (hardware), and 3) the mapping strategy of a DNN layer on the target hardware (mapping). That is, we can predict the efficiency (latency, energy, buffer requirement, etc.) of an accelerator when we have full parameters for those three factors, which can guide the DNN accelerator design for better efficiency. One critical requirement on the efficiency estimation is that it needs to be fast since the design space (e.g., 480 million valid designs in our hardware DSE even if we fix the target mapping and layer) is huge, and we need to query the efficiency of candidate designs in the search space when we search for an optimal design. How do we implement such a fast efficiency estimation framework that thoroughly considers all the parameters of the three factors that determine the efficiency of DNN accelerators?

Such demands led to the development of an analytical cost model instead of cycle-accurate simulators. Analytically, modeling the complex high-dimensional DNN accelerator design space over the three factors (DNN layer, hardware, and mapping) is challenging because it requires deep understanding of complex interaction of hardware components, mapping, and DNN layers. In particular, data reuse in scratchpad memory hierarchy in DNN accelerators is one of the key behaviors, which is critical for energy efficiency, thus the prime optimization target of DNN accelerators. Data reuse pattern is dictated by dataflow,[1] which are data/computation tile scheduling and spatial partitioning strategies without actual tile size as described in Figure 1 (a). To systematically and analytically model the data reuse for DNN accelerators' efficiency estimation, we need a precise and thorough description of mapping and a framework to analyze data reuse of a mapping on target hardware and the DNN layer.

Therefore, we propose a data-centric representation of mapping that enables precise descriptions of all the possible mappings in a concise and compiler-friendly manner. Leveraging the compiler-friendly format, we develop MAESTRO, a comprehensive cost-benefit analysis framework based on systematic data reuse analysis. As shown in Figure 1(b), MAESTRO receives the three factors—DNN layer, hardware, and mapping—as inputs and generates more than 20 estimated statistics including latency, energy, the number of buffer accesses, buffer size requirement, etc. We validated the performance statistics of MAESTRO against cycle-accurate RTL simulation results[5] and reported performance in a previous work[6] with

Table 1. The taxonomy of data reuse in DNN accelerators and implementation choices for each. We highlight implementation used in the example reuse patterns with red texts.

Data ReuseTaxonomy		Example Reuse Pattern in HW	Available HW Impmlementation Choices	
Reuse Type	Comm. Type			
Spatial (Across PEs)	Multicast	 PEs PE0 PE1 PE2 PE3 ⋯ 1 1 PE0 PE1 PE2 PE3 ⋯ 2 2 Time	 Fanout (e.g., Bus, Tree)	Store-and-Fwd (e.g., Systolic Array)
	Reduction	 PEs PE0 PE1 PE2 PE3 ⋯ 1 2 PE0 PE1 PE2 PE3 ⋯ 1 2 Time	 Fanin (e.g.,Reduction Tree)	Store-andFwd (e.g., Systolic Array)
Temporal (Over Time)	Multicast	 PEs PE0 PE1 PE2 PE3 ⋯ 1 PE0 PE1 PE2 PE3 ⋯ 1 Time	 Multiple reads from a buffer	
	Reduction	 PEs PE0 PE1 PE2 PE3 ⋯ 1 PE0 PE1 PE2 PE3 ⋯ 1 Time	 Multiple read-modify-write to a buffer	

1 Input/Filter Tile Index 1 Output Tile Index

the accuracy of 96.1% on average. MAESTRO provides fast cost-benefit estimation based on an analytical model, which took 493 ms to analyze the entire Resent50 layers[7] on a 256PE NVDLA-style[2] accelerator on a laptop with i9-9980H CPU with 16 GB of memory. MAESTRO supports arbitrary layer sizes and a variety of layer operations from state-of-the-art DNN models, which includes CONV1D, CONV2D, fully connected (FC) layer, depthwise separable convolution, up-scale convolution, etc.

DATA REUSE IN DNN ACCELERATORS

Data reuse is the key behavior in DNN accelerator that improves both latency and energy via reducing the number of remote buffer accesses (i.e., global buffer),[1,8] which is determined by dataflow. Data reuse opportunities exist when the dataflow assigns the same set of data tiles over consecutive time on the same PE (i.e., reuse in time) or across multiple PEs but not over consecutive time (i.e., reuse in space). We define those opportunities as temporal and spatial reuse opportunities. For example, in the example dataflow in Figure 1, output tiles (orange tiles) remain the same in time 0 and 1, which implies the temporal reuse opportunities. Within time 1, as the spatial partitioning example in Figure 1 shows, input tile 3 is mapped on all the PEs, which implies the spatial reuse opportunities.

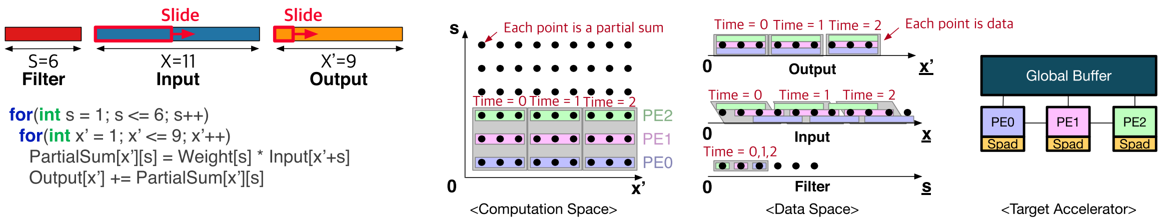
Dataflow implies data reuse opportunities, and we can categorize data reuse in DNN accelerators into four types (data reuse taxonomy), which we summarize in Table 1. Each data reuse type requires proper hardware support to exploit the data reuse opportunity as actual data reuse. We discuss those four reuse types grouped in communication type as follows:

Spatial/Temporal Multicast. When the spatial/temporal reuse opportunities are in input tensors (i.e., filter and input activation), the reused data can be multicasted to multiple PEs (spatial reuse) or over time (temporal reuse). The examples in Table 1 show such a pattern based on fanout NoC (spatial multicast), which delivers data to multiple PEs at the same time, and buffer (temporal multicast).

In the spatial multicast example, tiles 1 and 2 are delivered to PE1 and PE2 at the same time leveraging the multicast capability of fanout hardware. Alternatively, store-and-forward style implementation such as systolic arrays is available with tradeoff of hardware cost and latency. In the temporal multicast example, the same data tile appears over time in the same PE (PE1). That is, we send the data to the future for reuse in the future (i.e., store the data in a buffer and read it in the future). Therefore, temporal multicast, which is reading the same stored data over time, requires a buffer, as shown in Table 1.

Spatial/Temporal Reduction. When the spatial reuse opportunities are in the output activation tensor, the reuse pattern in hardware is spatial reduction, which accumulates partial outputs (or, partial sums) for an output across multiple PEs. The example in Table 1 shows an example reuse pattern based on store-and-forward hardware. We observe that the output tiles 1 and 2 are moving to the next PE over time, which illustrates pipelined accumulation to the right direction assuming that PEs are receiving new operands from above (i.e., a row of a systolic array). Alternatively, fanin hardware such as reduction tree can support the spatial reduction.

In contrast, the temporal reuse opportunities imply that we compute partial sums over time and accumulate them within the same location. This type of reuse requires a buffer since



```
for(int s = 1; s <= 6; s++)
  for(int x' = 1; x' <= 9; x'++)
    PartialSum[x'][s] = Weight[s] * Input[x'+s]
    Output[x'] += PartialSum[x'][s]
```

(a) An Example CONV1D Operation

(b) An Example Mapping of CONV1D

Figure 2. Example CONV1D operation and mapping of the example on an accelerator. We represent the mapping in both computation and data space, where each point corresponds to a partial sum and a data, respectively. We use 1-based indices in this example.

intermediate results need to be stored and read again in the future, which effectively indicates multiple read-modify-write to a buffer. The example in Table 1 shows such a reuse pattern, where the output tile 1 appears at the same PE over time.

To identify the reuse opportunities in arbitrary mappings, we need a precise representation of mapping and systematically infer data reuse from the description. For those two goals, we present a data-centric representation of mapping, which is concise and compiler friendly.

DESCRIBING MAPPINGS

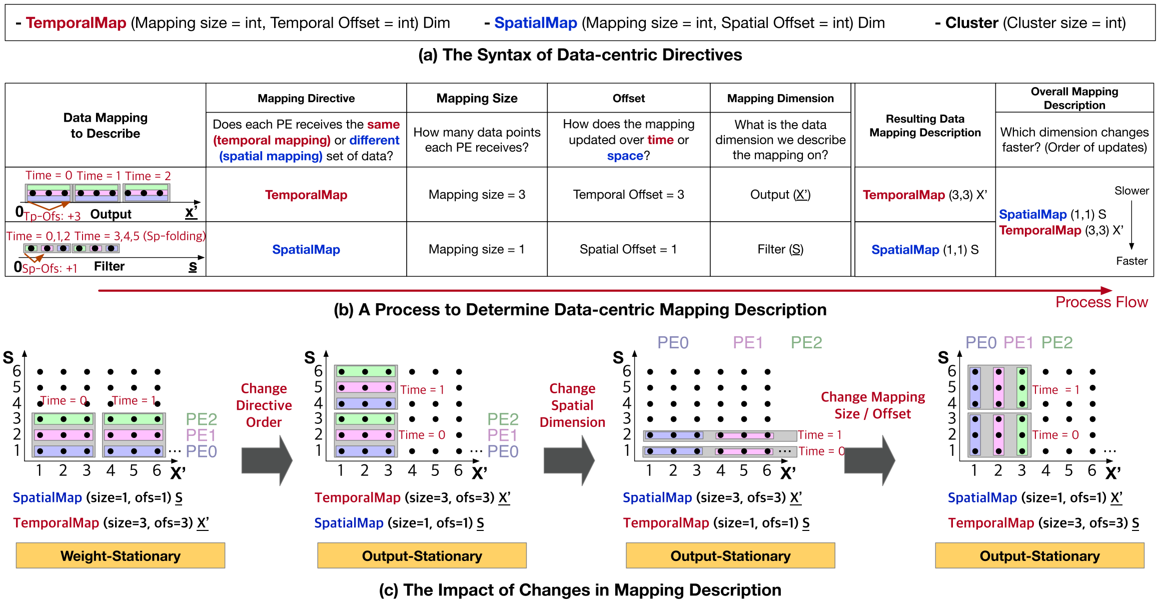
We use a CONV1D operation described in Figure 2 as an example operation to introduce our mapping description. As described in Figure 2, CONV1D operation can be understood as a sliding window operation of a filter vector on a input vector, where individual multiplication results within a filter window are accumulated to generated one output value in the output vector. When we project the loop indices in the loop nest in Figure 2(a), we obtain computation space in Figure 2(b) where loop indices are on each axis, and partial sums are projected in the plane. We also construct data space of each vector as shown in Figure 2(b), where the corresponding data index is on the axis. Note that the data index is not the same as the loop index (e.g., the input data index $\underline{x}$ is computed using loop indices $x'+s$). Therefore, we denote data indices using underlined index in this example. Note that output and filter indices $\underline{x}'$ and $\underline{s}$ are identical to the loop indices x' and s in this simple example operation.

We show an example of mapping on three-PE accelerator in computation and data space in Figure 2(b). In this example mapping, we map three partial sum computation to each PE, and each PE collaboratively compute partial outputs (accumulated partial sums) on the same set of outputs. When the PE array finishes computation in a tile (time=0 in the example), the PE array receives the next computation tile (time=1 in the example). The next computation tile is in the direction of loop index x′. We project the same mapping on the data space as shown in Figure 2, using the array subscripts in the loop nest of CONV1D operation in Figure 2(a). That is, partial sum at (x,′ s) requires weight at s, input at x′+s, and output at x,′ as shown in the loop body of in Figure 2(a). In the example, we observe that the data space explicitly shows data reuse behavior; mapped filter values do not move over time, which implies that the example mapping is based on a weight-stationary style dataflow. This implies that inferring data reuse can be significantly simplified when we describe mapping in the data space, which can facilitate a fast analysis framework of DNN accelerator's efficiency.

Motivated by the observation, we introduce data-centric mapping directives that directly describe the mapping in data space.

Data-centric mapping directives

We introduce three data-centric mapping directives in Figure 3(a). Temporal and spatial map directives describe data mapping that changes in time and space (PEs), respectively. That is, temporal map corresponds to a normal for loop in loop nest while spatial map corresponds to a parallel for loop. Those two mapping



- **TemporalMap** (Mapping size = int, Temporal Offset = int) Dim - **SpatialMap** (Mapping size = int, Spatial Offset = int) Dim - **Cluster** (Cluster size = int)

(a) The Syntax of Data-centric Directives

Data Mapping to Describe	Mapping Directive	Mapping Size	Offset	Mapping Dimension	Resulting Data Mapping Description	Overall Mapping Description
	Does each PE receives the same (temporal mapping) or different (spatial mapping) set of data?	How many data points each PE receives?	How does the mapping updated over time or space?	What is the data dimension we describe the mapping on?		Which dimension changes faster? (Order of updates)
Time = 0 Time = 1 Time = 2 0 Tp-Ofs: +3 Output X'	TemporalMap	Mapping size = 3	Temporal Offset = 3	Output (X')	TemporalMap (3,3) X'	Slower SpatialMap (1,1) S TemporalMap (3,3) X' Faster
Time = 0,1,2 Time = 3,4,5 (Sp-folding) 0 Sp-Ofs: +1 Filter S	SpatialMap	Mapping size = 1	Spatial Offset = 1	Filter (S)	SpatialMap (1,1) S	

Process Flow
(b) A Process to Determine Data-centric Mapping Description
S
X'
Time = 0
Time = 1
PE0
PE1
PE2
SpatialMap (size=1, ofs=1) S
TemporalMap (size=3, ofs=3) X'
Weight-Stationary
Change Directive Order
TemporalMap (size=3, ofs=3) X'
SpatialMap (size=1, ofs=1) S
Output-Stationary
Change Spatial Dimension
SpatialMap (size=3, ofs=3) X'
TemporalMap (size=1, ofs=1) S
Output-Stationary
Change Mapping Size / Offset
SpatialMap (size=1, ofs=1) X'
TemporalMap (size=3, ofs=3) S
Output-Stationary
(c) The Impact of Changes in Mapping Description

Figure 3. Introductory example of data-centric directives. (a) Syntax of data-centric directives. (b) Semantics of two mapping directives based on an example description process on the example CONV1D mapping in Figure 2. (c) Capability of data-centric mapping directives that can describe a variety of mapping styles.

directives take three parameters: Mapping size, offset, and dimension. The mapping size specifies the number of data points (in tensors, mapping size in the target dimension since a mapping constructs a high-dimensional volume) mapped on each PE. The offset describes how the mapping is updated over time on temporal map and space on spatial map. Cluster directive specifies the hierarchical organization of PEs, which enables us to explore multiple parallel dimensions in a mapping.

To understand the syntax and semantics of data-centric mapping directives, in Figure 3, we provide an example process to determine a corresponding data mapping description of the example mapping in Figure 2(b). We omit the input tensor because input tensor data mapping can be easily inferred from the mapping of output and filter. We first determine if the mapping is in time or space by checking the mapped data are the same or different (i.e., parallelization) across PEs. Next, we check the number of data points mapped on each PE to determine the mapping size, which are three and one for output and filter, respectively, in the example. To determine the offset parameter, we check the temporal and spatial offset on temporal and spatial map, respectively. For example, for output vector in Figure 3(b), we observe that the starting index of mapping changes over time 3, which implies that the temporal offset is 3. For filter vector, we observe that the starting index of mapping for each PE changes by 1, which implies that the spatial offset is 1. Note that spatial map can also involve temporal aspect as the mapping on the filter vector in Figure 3(b); after processing all the computation that involves the first data tile on filter, the data tile will move on to the next position. This happens when the number of PEs is not sufficient to cover entire spatially mapped dimension (also known as spatial folding), and an implicit temporal offset of (spatial offset) × (number of PEs) is applied. Finally, we write the dimension on which we describe the data mapping, then we obtain the data-centric mapping description of each data mapping, as shown in the resulting data mapping description column in Figure 3(b). To specify the entire example mapping, we need to specify the order of changes in data tile between output and filter vectors. Since filter is updated in a slower manner, we place the data mapping description of filter above, and write that of output below, like we specify the update order in loop nest (outermost loop index changes slower).

Capability of Mapping Directives

Using the data-centric directives, we can describe a variety of mappings if it maps consecutive data points in a regular manner (i.e., affine loop subscripts when described in a loop nest representation). Figure 3(c) shows the capability of the data-centric directive by showing the changes in the resulting mapping when we update the base representation we obtained in Figure 3(b). When we change the directive order, we describe a different order of data tile update in dimensions. This effectively changes the stationary vector from weight to output, which changes the temporal data reuse opportunities. When we change the spatial dimension, then we exploit the parallelism in a different dimension, as the third example in Figure 3(b) and (c) shows. Finally, if we change the mapping size (we accordingly update the offset to keep the description legal), we change the amount of mapped filter and output, as shown in Figure 3(c) and (d).

Based on the fact that data reuse is explicit in data dimension and the capability of data-centric directives, we implement an analytical cost-benefit analysis framework for DNN accelerators, MAESTRO. We discuss a high-level overview of MAESTRO next and discuss insights from the case studies we performed based on MAESTRO next.

ANALYTICAL COST MODEL

Based on the data-centric directives we discussed, we built a cost-benefit analysis framework that considers all of the three factors—DNN layers, hardware, and mapping—with precise modeling of data reuse. MAESTRO consists of five preliminary engines: Tensor, cluster, reuse, performance analysis, and cost analysis. In the article, we focus on the high-level idea without details such as edge case handling, multiple layers, and multiple level hierarchy, etc. We present implementation details in our web page and open-source repository.* We validated MAESTRO's performance model against RTL simulation and reported processing delay of two accelerators—MAERI[5] and Eyeriss[6] when running VGG16 and AlexNet, respectively. The latency estimated by MAESTRO are within 3.9% absolute error of the cycle-accurate RTL simulation and reported processing delay[6] on average.

CASE STUDIES

With MAESTRO, we perform deeper case studies about the costs-benefit tradeoff of various mappings when applied to different DNN operations. We evaluate five distinct mapping styles listed in Figure 4(a) in the "Case Study I: The Impact of Mapping Choices" section and the preference of each mapping to different DNN operators. For energy estimation, we multiply activity counts with base energy values from Cacti[13] simulation (28 nm, 2 kB L1 scratchpad, and 1 MB shared L2 buffer). We also present distinct design space of an early layer (wide and shallow) and a late layer (narrow and deep) to show the dramatically different hardware preference of different DNN layers and mapping in the "Case Study II: Hardware Design-Parameters and Implementation Analysis" section.

Case Study I: The Impact of Mapping Choices

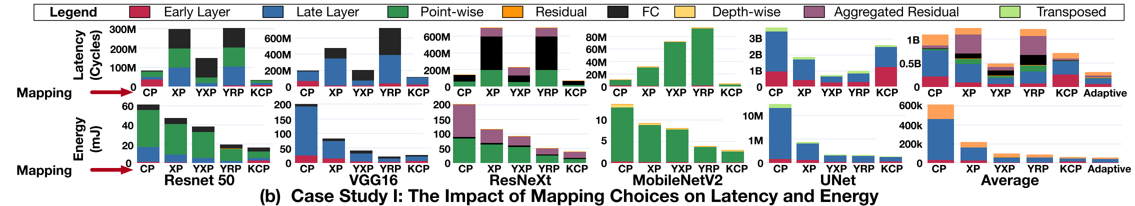
Figure 4(b) shows the DNN-operator granularity estimation of latency and energy of each mapping across five state-of-the-art DNN models listed in the "Case Studies" section. Note that this should be considered a comparison of mapping—not of actual designs, which can contain several low-level implementation differences, e.g., custom implementations of logic/memory blocks, process technology, etc. We observe that KC-P style mapping provides overall low latency and energy. However, the energy efficiency in VGG16 is worse than YR-P (Eyeriss[1] style) mapping, and the latency is worse than YX-P (Shidiannao[14] style) mapping in UNet. This is based on the different preference toward mapping of each DNN operator. YX-P provides short latency to segmentation networks like UNet, which has wide activation (e.g., 572 × 572 in the input layer) and recovers the original activation dimension at the end via up-scale convolution (e.g., transposed convolutions). Such a preference to the YX-P style is mainly based on its parallelization strategy: It exploits parallelism over both of row and column dimensions in

*https://maestro.ece.gatech.edu/

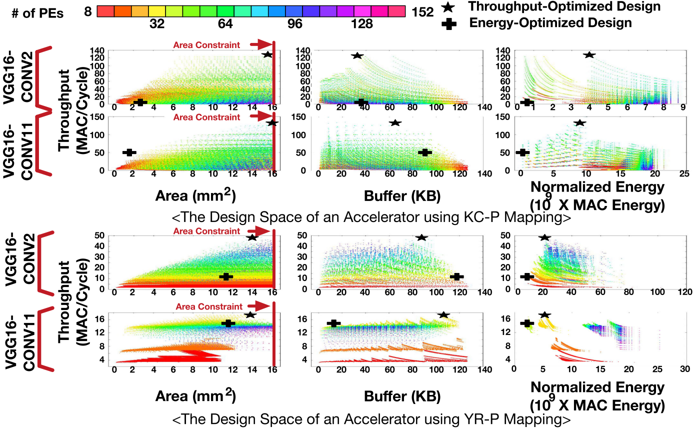
Partitioning Strategy	Mapping	Characteristics
C-Partitioned (C-P)	TemporalMap (1,1) K TemporalMap (Sz(R),1) Y TemporalMap (Sz(S),1) X TemporalMap (Sz(R),Sz(R)) R TemporalMap (Sz(S),Sz(S)) S SpatialMap (1,1) C	- Large spatial reduction opportunities (High output activation reuse) - Low input activation/filter reuse - Input channel (C) parallelism - No local reuse
X-Partitioned (X-P)	TemporalMap (1,1) K TemporalMap (1,1) C TemporalMap (Sz(R),Sz(R)) R TemporalMap (Sz(S),Sz(S)) S TemporalMap (Sz(R),1) Y SpatialMap (Sz(S),1) X	- High temporal reuse of filter - Spatial reuse opportunities (via halo in input activation) - Input column (X) parallelism - Weight-stationary
YX-Partitioned (YX-P)	TemporalMap (1,1) K SpatialMap (Sz(R),1) Y TemporalMap (8+Sz(S)-1,8) X TemporalMap (1,1) C TemporalMap (Sz(R),Sz(R)) R TemporalMap (Sz(S),Sz(S)) S Cluster(8) SpatialMap (Sz(S),1) X	- High temporal reuse of filter - Better spatial reuse opportunities over X-P (via 2D halo in input activation) - 2D activation (X and Y) parallelism - Output-stationary - Motivated by Shi-diannao [14]

Partitioning Strategy	Mapping	Characteristics
YR-Partitioned (YR-P)	TemporalMap (2,2) C TemporalMap (2,2) K SpatialMap (Sz(R),1) Y TemporalMap (Sz(S),1) X TemporalMap (Sz(R),Sz(R)) R TemporalMap (Sz(S),Sz(S)) S Cluster(Sz(R)) SpatialMap (1,1) Y SpatialMap (1,1) R	- High temporal reuse of input activation and filter - Spatial reduction opportunities (spatial reuse of output activation) - Activation row (Y) and filter column (S) parallelism - Row-stationary - Motivated by Eyeriss [1]
KC-Partitioned (KC-P)	SpatialMap (1,1) K TemporalMap (64,64) C TemporalMap (Sz(R),Sz(R)) R TemporalMap (Sz(S),Sz(S)) S TemporalMap (Sz(R),1) Y TemporalMap (Sz(S),1) X Cluster(64) SpatialMap (1,1) C	- Spatial reuse of input activation - High spatial reduction factor (64-way) on input channel (C) - Input/output channel (C and K) parallelism - Weight-stationary - Motivated by NVDLA [2]

(a) Mappings Used in Case Study I



(b) Case Study I: The Impact of Mapping Choices on Latency and Energy



<The Design Space of an Accelerator using KC-P Mapping>

<The Design Space of an Accelerator using YR-P Mapping>

(c) Case Study II: Design Space Characterization

Configs & Stats \ Design Points		Reference	Small Bandwidth	No Multicast	No Spatial Reduction
Number of PEs		56	56	56	56
NoC Bandwidth (GBps)		40	24	40	40
Spatial Reuse Support	Mult. Cast	Yes	Yes	No	Yes
	Redn.	Yes	Yes	Yes	No
Throughput (MACs/Cycle)		48.6	34.54	33.39	32.05
Energy (X GMACs)		5.26	5.26	7.56	7.77

(d) Example Design Points of KC-P Mapping-based Accelerators on VGG16-CONV2

Figure 4. Summary of case studies. (a) List of mappings used in case study I. (b) Results of the case study I. Top and bottom rows present latency and energy, respectively. We apply 256 PEs and 32 GBps NoC bandwidth. We use five different DNN models; Resnet50,[7] VGG16,[9] ResNeXt50,[10] MobileNetV2,[11] and UNet.[12] The right-most column presents the average results across models for each DNN operator type and the adaptive mapping case. We compare the number of input channels and the input activation height to identify early and late layers (If C > Y, late layer. Else, early layer). (c) Design space of KC-P and YR-P-based accelerators. We highlight the design space of an early and a late layer to show their significantly different hardware preference. We apply area/power constraints based on Eyeriss[6] to the DSE. The color of each data point indicates the number of PEs. We mark the throughput- and energy-optimized designs using stars and crosses. (d) The impact of multicast capability, bandwidth, and buffer size. Design points are selected from the upper-most design space in (c). The name of design points refer to the differences from the throughput-optimal reference point. Dark rows represent the efficiency of the selected design point.

activation. The energy efficiency of YR-P mapping in VGG16 is based on its high reuse factor (the number of local accesses per fetch) in early layers. The YR-P mapping has 5.8× and 15.17× higher activation and filter reuse factors, respectively, in early layers. However, in late layers, the reuse factors of YR-P and KC-P mapping are almost similar (difference < 11%), so the KC-P mapping provides similar energy efficiency as YR-P in these cases. This can also be observed in the late layer (blue) bars in Figure 4(b) bottom-row plots.

The diverse preference to mappings of different DNN operators motivates us to employ optimal mapping for each DNN operator type. We refer such an approach as adaptive mapping and present the benefits in the right-most column of Figure 4(b), the average case analysis across entire models in the DNN operator granularity. By employing the adaptive approach, we could observe a potential 37% latency and 10% energy reduction. Such an optimization opportunity can be exploited by flexible accelerators like Flexflow[15] and MAERI[5] or via heterogeneous accelerators that employ multiple subaccelerators with various mapping styles in a single DNN accelerator chip.

Case Study II: Hardware Design-Parameters and Implementation Analysis

Using MAESTRO, we implement a hardware design space exploration (DSE) tool that searches four hardware parameters (the number of PEs, L1 buffer size, L2 buffer size, and NoC bandwidth) optimized for either energy efficiency, throughput, or energy-delay-product (EDP) within given hardware area and power constraints. The DSE tool receives the same set of inputs as MAESTRO with hardware area/power constraints and the area/power of building blocks synthesized with the target technology. For the cost of building blocks, we implement float/fixed point multiplier and adder, bus, bus arbiter, and global/local scratchpad in RTL and synthesis them using 28-nm technology. For bus and arbiter cost, we fit the costs into a linear and quadratic model using regression because bus cost increases linearly and arbiter cost increases quadratically (e.g., matrix arbiter).

Using the DSE tool, we explore the design space of KC-P and YR-P mapping accelerators. We set the area and power constraint as 16 mm^2 and 450 mW, which is the reported chip area and power of Eyeriss.[6] We plot the entire design space we explored in Figure 4(c). Whether an accelerator can achieve peak throughput depends on not only the number of PEs but also NoC bandwidth. In particular, although an accelerator has sufficient number of PEs to exploit the maximum degree of parallelism a mapping allows, if the NoC does not provide sufficient bandwidth, the accelerator suffers a communication bottleneck in the NoC. Such design points can be observed in the area-throughput plot in Figure 4(c). YR-P mapping requires low NoC bandwidth so it does not show the same behavior as KC-P mapping. However, with more stringent area and power constraints, YR-P mapping will show the same behavior.

During DSE runs, MAESTRO reports buffer requirements for each mapping and the DSE tool places the exact amount buffers MAESTRO reported. Contrary to intuition, larger buffer sizes do not always provide high throughput, as shown in buffer-throughput plots in Figure 4 (plots in the second column). The optimal points regarding the throughput per buffer size are in the top-left region of the buffer-throughput plots. The existence of such points indicates that the tiling strategy of the mapping (mapping sizes in our directive representation) significantly affects the efficiency of buffer use. We observe that the throughput-optimized designs have a moderate number of PEs and buffer sizes, implying that hardware resources need to be distributed not only to PEs but also to NoC and buffers for high PE utilization. Likewise, we observe that the buffer amount does not directly increase throughput and energy efficiency. These results imply that all the components are intertwined, and they need to be well-balanced to obtain a highly efficient accelerator.

We also observe the impact of hardware support for each data reuse type, discussed in Table 1. Figure 4(d) shows such design points found in the design space of KC-P mapping on VGG16-conv2 layer presented in the first row of Figure 4(c). The reference design point is the throughput-optimized design represented as a star in the first row of Figure 4(c). When bandwidth gets smaller, the throughput significantly drops, but energy remains similar. However, the lack of spatial multicast or reduction support resulted in approximately 47% energy increase, as the third and fourth design points shows.

CONCLUSION

Fast modeling of cost-benefit space of DNN accelerators is critical for automated optimization tools since the design space is huge and high dimensional based on hundreds of DNN model, hardware, and mapping parameters. In this article, we presented a methodology to enable fast cost-benefit estimation of a DNN accelerator on a given DNN model and mapping, which consists of a compiler-friendly data-centric representation of mappings and an analytical cost-benefit estimation framework that exploits the explicit data reuse in data space in data-centric representations. To analytically estimate the costs and benefits, we demystify data reuse in hardware and required hardware support and apply the observation into the analytical cost-benefit estimation framework, MAESTRO.

Using MAESTRO, we show that no single mapping and no single hardware is ideal for all the DNN layers, which implies the complexity of the DNN accelerator design space. Using hardware design space exploration framework we implemented using MAESTRO, we also show that hardware features can significantly impact the throughput and energy. Those cases show that the capability of MAESTRO for various analysis problems on DNN accelerator design space. In addition to the case studies we performed, MAESTRO also facilitates many other optimization (e.g., neural architecture search specialized for a target accelerator, mapping search for a target accelerator, etc.) frameworks based on its speed and accuracy, which will lead to broad impact on various areas (DNN model design, compiler, architecture, etc.) in the DNN accelerator domain.

> Using MAESTRO, we show that no single mapping and no single hardware is ideal for all the DNN layers, which implies the complexity of the DNN accelerator design space. Using hardware design space exploration framework we implemented using MAESTRO, we also show that hardware features can significantly impact the throughput and energy.

REFERENCES

1. Y.-H. Chen, J. Emer, and V. Sze, “Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks,” in *Proc. Int. Symp. Comput. Archit.*, 2016, pp. 367–379.
2. “Nvdla deep learning accelerator,” 2017. [Online]. Available: http://nvdla.org.
3. A. Parashar *et al.*, “Scnn: An accelerator for compressed-sparse convolutional neural networks,” in *Proc. Int. Symp. Comput. Archit.*, 2017, pp. 27–40.
4. N. P. Jouppi *et al.*, “In-datacenter performance analysis of a tensor processing unit,” in *Proc. IEEE Int. Symp. Comput. Archit.*, 2017, pp. 1–12.
5. H. Kwon, A. Samajdar, and T. Krishna, “Maeri: Enabling flexible dataflow mapping over DNN accelerators via reconfigurable interconnects,” in *Proc. Int. Conf. Archit. Support Program. Lang. Oper. Syst.*, 2018, pp. 461–475.
6. Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, “Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks,” *IEEE J. Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, Jan. 2017.
7. K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.
8. A. Parashar *et al.*, “Timeloop: A systematic approach to DNN accelerator evaluation,” in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw.*, Mar. 2019, pp. 304–315.
9. K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” in *Proc. Int. Conf. Learn. Representations*, 2015. [Online]. Available: https://iclr.cc/archive/www/doku.php%3Fid=iclr2015:accepted-main.html
10. S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, “Aggregated residual transformations for deep neural networks,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 1492–1500.
11. M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. Chen, “MobileNetV2: Inverted Residuals and Linear Bottlenecks,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 4510–4520.
12. O. Ronneberger, P. Fischer, and T. Brox, “U-net: Convolutional networks for biomedical image segmentation,” in *Proc. Int. Conf. Med. Image Comput. Comput.-Assisted Intervention*, 2015, pp. 234–241.
13. N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi, “Cacti 6.0: A tool to model large caches,” *HP Laboratories*, vol. 27, p. 28, 2009.
14. Z. Du *et al.*, “Shidiannao: Shifting vision processing closer to the sensor,” in *Proc. Int. Symp. Comput. Archit*, 2015, pp. 92–104.

15. W. Lu, G. Yan, J. Li, S. Gong, Y. Han, and X. Li, "Flexflow: A flexible dataflow accelerator architecture for convolutional neural networks," in *Proc. Int. Symp. High Perform. Comput. Archit.*, 2017, pp. 553–564.

Hyoukjun Kwon is currently working toward the Ph.D. degree in the College of Computing, Georgia Institute of Technology. His research interest includes communication-centric and flexible accelerator design and modeling mappings on spatial accelerators. Kwon received B.S. degrees in environmental materials science and in computer science and engineering from Seoul National University. He is a student member of IEEE. Contact him at hyoukjun@gatech.edu.

Prasanth Chatarasi is a senior Ph.D. student advised by Prof. Vivek Sarkar and Dr. Jun Shirako in the School of Computer Science, Georgia Institute of Technology. His research focuses on advancing compiler optimizations for high-performance applications on general-purpose and domain-specific parallel architectures. In the past, he focused on enhancing traditional compilation techniques for both sequential and explicitly parallel programs for performance optimizations and debugging on general-purpose architectures. Contact him at cprasanth@gatech.edu.

Vivek Sarkar is a Professor and the Stephen Fleming Chair for Telecommunications in the College of Computing at Georgia Institute of Technology, where he conducts research in multiple aspects of software for parallel computing. He is a Fellow of ACM and IEEE. Contact him at vsarkar@gatech.edu.

Tushar Krishna is an Assistant Professor in the School of Electrical and Computer Engineering, Georgia Institute of Technology, where he also holds the ON Semiconductor Junior Professorship. His research interests include computer architecture, on-chip interconnection networks, and deep learning accelerators. Krishna received the Ph.D. degree in electrical engineering and computer science from Massachusetts Institute of Technology. He received the NSF CRII Award in 2018. He is a member of IEEE and ACM. Contact him at tushar@ece.gatech.edu.

Michael Pellauer is a Senior Research Scientist at NVIDIA. His research interests are building domain specific accelerators, with a special emphasis on deep learning and sparse tensor algebra. Pellauer received the Ph.D. degree from Massachusetts Institute of Technology, the Masters degree from Chalmers University of Technology, and the Bachelor's degree from Brown University. Contact him at mpellauer@nvida.com.

Angshuman Parashar is a Senior Research Scientist at NVIDIA. His research interests are in building, evaluating, and programming spatial and data-parallel architectures, with a present focus on automated mapping of machine learning algorithms onto architectures based on explicit decoupled data orchestration. Parashar received the Ph.D. degree in computer science and engineering from the Pennsylvania State University (2007), and the B.Tech. degree in computer science and engineering from the Indian Institute of Technology, Delhi, India (2002). Contact him at aparashar@nvidia.com.