Marvel: A Data-Centric Approach for Mapping Deep Learning Operators on Spatial Accelerators

PRASANTH CHATARASI and HYOUKJUN KWON, Georgia Institute of Technology
ANGSHUMAN PARASHAR and MICHAEL PELLAUER, NVIDIA
TUSHAR KRISHNA and VIVEK SARKAR, Georgia Institute of Technology

A spatial accelerator's efficiency depends heavily on both its mapper and cost models to generate optimized mappings for various operators of DNN models. However, existing cost models lack a formal boundary over their input programs (operators) for accurate and tractable cost analysis of the mappings, and this results in adaptability challenges to the cost models for new operators. We consider the recently introduced Maestro Data-Centric (MDC) notation and its analytical cost model to address this challenge because any mapping expressed in the notation is precisely analyzable using the MDC's cost model.

In this article, we characterize the set of input operators and their mappings expressed in the MDC notation by introducing a *set of conformability rules*. The outcome of these rules is that any loop nest that is perfectly nested with affine tensor subscripts and without conditionals is conformable to the MDC notation. A majority of the primitive operators in deep learning are such loop nests. In addition, our rules enable us to automatically translate a mapping expressed in the loop nest form to MDC notation and use the MDC's cost model to guide upstream mappers. Our conformability rules over the input operators result in a *structured mapping space* of the operators, which enables us to introduce a mapper based on our *decoupled off-chip/on-chip* approach to accelerate mapping space exploration. Our mapper decomposes the original higher-dimensional mapping space of operators into two lower-dimensional off-chip and on-chip subspaces and then optimizes the off-chip subspace followed by the on-chip subspace. We implemented our overall approach in a tool called *Marvel*, and a benefit of our approach is that it applies to any operator conformable with the MDC notation. We evaluated Marvel over major DNN operators and compared it with past optimizers.

CCS Concepts: • **Hardware → Hardware accelerators**; • **Software and its engineering → Compilers**; • **Computing methodologies → Neural networks;**

Additional Key Words and Phrases: Deep learning (spatial) accelerators, compilers, mappers/optimizers

ACM Reference format:
Prasanth Chatarasi, Hyoukjun Kwon, Angshuman Parashar, Michael Pellauer, Tushar Krishna, and Vivek Sarkar. 2021. Marvel: A Data-Centric Approach for Mapping Deep Learning Operators on Spatial Accelerators. *ACM Trans. Arch. Code Optim.* 19, 1, Article 6 (December 2021), 26 pages.
https://doi.org/10.1145/3485137

6

Prasanth Chatarasi is now at IBM Research. Hyoukjun Kwon is now at Facebook.
This work was supported by NSF Awards 1755876, 1909900, and 1822939, and also the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA-0003525.
Authors' addresses: P. Chatarasi, H. Kwon, T. Krishna, and V. Sarkar, Georgia Institute of Technology, North Ave NW, Atlanta, GA 30332; emails: prasanth@ibm.com, hyoukjunkwon@fb.com, tushar@ece.gatech.edu, vsarkar@gatech.edu; A. Parashar and M. Pellauer, NVIDIA Corporation, 2 Technology Park Drive Floor 3 Westford, MA 01886; emails: {aparashar, mpellauer}@nvidia.com.

1544-3566/2021/12-ART6 $15.00
https://doi.org/10.1145/3485137

1 INTRODUCTION

Deep learning (DL) is a fundamental technology for many emerging applications, such as autonomous driving [4], translation [52], and image classification [42], with accuracy close to, and even surpassing, that of humans [15, 21, 49]. To significantly improve the performance in terms of latency and energy, specialized hardware accelerators for DNN inference are being developed and deployed [1, 10, 11, 33, 53]. Some of the most popular examples are systolic arrays such as TPU [20], xDNN [53], and RAPID [14], and advanced forms such as NVDLA [33], Eyeriss [7], ShiDianNao [13], and MAERI [26]. The primary architectural features that distinguish these custom "spatial" accelerators from CPUs and GPUs are (1) parallelism using hundreds to thousands of **processing elements (PEs)**, (2) fast **network-on-chip (NoC)** connecting these, and (3) use of private/shared scratchpad memories for data reuse. They achieve high throughput by exploiting parallelism over the PEs and energy efficiency by maximizing data reuse within the PE array via direct data forwarding between PEs and the use of scratchpad memories [5, 7, 20, 29, 33, 35, 46, 56].

In general, compilers for DL consist of two major components: (1) graph optimizer performing graph-oriented transformations (e.g., node fusion) over an input model computation graph, and (2) kernel optimizer loop-oriented transformations (e.g., tiling, reordering) to map and generate code for each node of the modified graph onto the accelerator's compute and memory resources. Mappers belong to the kernel optimizers of DL compilers, and the role of mappers is to find optimized schedules. Optimized mappers optimizing various DNN operators are necessary during compile-time for ML programmers, and design-time for computer architects to understand reuse and data movement behaviors to design a new accelerator, as shown in Figure 1.

A major difference between the mappers for spatial accelerators and CPUs/GPUs is the need for an "accurate" analytical cost model that can estimate close to real-world values for a given mapping of a kernel (operator) on an accelerator configuration [24, 34]. This is because edge conditions can lead to a multiplicative slowdown if not accounted for. As a simple example, suppose a convolution has 17 input channels and 32 output channels, whereas hardware has 16 PEs. Suppose mapping A parallelizes across the input channels, whereas mapping B parallelizes across output channels. Mapping A will require two folds (rounds of execution) with 94% under-utilization in the second fold, whereas mapping B will never have any under-utilization. An approximate analytical model that ignores the second fold of Mapping A due to rounding will under-estimate the runtime of this mapping by 2×. Such roundings over multiple iterations can lead to a significant difference from real values.

Recently, multiple analytical cost models [12, 24, 34, 54] have been proposed to estimate execution time and energy efficiency of mappings over spatial accelerators close to the real-world simulation/execution. These mappings are often expressed in the form of *loop nests*, a syntax that resembles a simple imperative programming language with explicit parallelism. Many cost models such as Timeloop [34], DMazeRunner [12], and Interstellar [54] are developed over the loop nest description of mappings. The loop nest syntax is very generic and can help accelerator architects or compilers in expressing a wide range of mappings, *but the underlying cost models may not be able to estimate costs for all possible mappings that can be expressible in loop nests.*[1] For example, a mapping for the parametric multi-step LSTM operator may include loop skewing transformation to exploit pipelined (wave-front) parallelism [3], but it is not clear if the cost models mentioned previously can precisely reason and estimate costs for such mappings. The existing cost models do not have a formal boundary over operators for precise cost analysis of the operator mappings.

[1]The generic loop nest forms encompass various control flow structures and data-access patterns that make it harder for the cost models to precisely reason (more details can be found in Section 7).

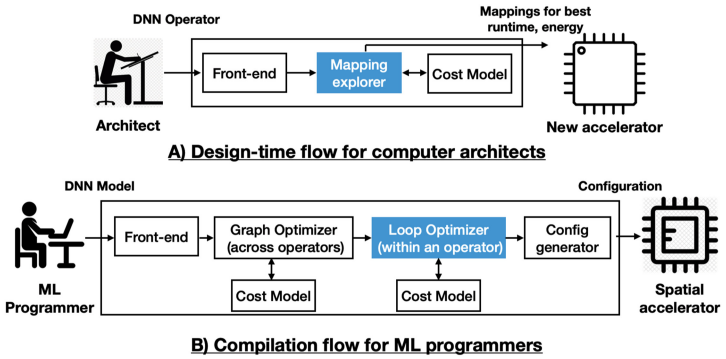

Fig. 1. Overview of the design-time flow for computer architects developing new accelerators, and the compilation flow for ML programmers leveraging the accelerators. The scope of this work is the mapping explorer and the loop optimizer in the preceding diagram.

None of the prior work attempts at defining a formal boundary, and having no such boundaries can bring adaptability challenges to these cost models in the kernel optimizers (challenge 1).

Besides the lack of the existing cost model's formal boundaries, searching for optimal mappings is challenging because of the massive space of the operator's legal mappings on the accelerator configurations. For example, there are more than 10^{19} valid mappings for the convolution operator on average for mapping ResNet50 [17] and MobileNetV2 [43] on a representative DNN edge accelerator. On one side, much of the prior work targeted hardware with limited capabilities (e.g., mRNA [57] for the MAERI accelerator, TVM extensions [30] for the VTA GEMM accelerator, and DeepTools [51] for the RAPID AI accelerator), which makes them not directly applicable to generic spatial accelerators. On another side, prior work on generic templated spatial accelerators fixed certain aspects of the mapping space such as choice of parallel loops and loop orders [12, 29, 32, 54, 56], and such limited exploration can limit the possibilities in achieving the accelerator's peak performance for diverse operators. To the best of our knowledge, Timeloop [34] is the only framework that considers all aspects of a mapping for generic templated spatial accelerators. However, it employs either an exhaustive linear search or a random sampling-based heuristic to explore the search space. *Hence, approaches supporting generic templated spatial accelerators and exploring all possible mappings suffer from a combinatoric explosion in the size of mapping space (challenge 2).*

The key contributions of our work addressing the preceding two challenges are described next.

Conformable DNN operators. To address the first challenge, we consider the recently introduced **Maestro Data-Centric (MDC)** notation [24] for expressing mappings onto generic templated spatial architectures. MDC is promising because any mapping expressed in the notation is precisely analyzable using the MDC's cost model [25]. Instead of proposing an approach for validating an input mapping in the MDC notation, we slightly take a different direction. We introduce a set of conformability formal rules (Section 3) where if any operator satisfies/conforms to those rules, all possible mappings of the operator are expressible in the MDC notation. As an example, Table 1 lists the conformability of the popular DNN operators with the MDC notation. Furthermore, all the primitive operators identified from profiling MLPerf benchmark suite [38], VGG16 [48], and AlexNet [22] models are MDC conformable, and these operators did not require any rewriting to make them conformable with the MDC notation.

The MDC notation explicitly requires defining data movement aspects of a mapping, instead of inferring in loop nest notation, which makes estimating execution time and energy efficiency relatively faster. However, since MDC notation is relatively new, it can be challenging for

architect/compiler experts to specify mappings with explicit data movement. Hence, we introduce a *transformation* (Section 4) that translates a mapping specified in the loop nest form to the MDC notation, and this can also be used for mapping space exploration.

Mapping space exploration. Our conformability rules over the input operators result in a *structured mapping space* of the operators, and this enables us to introduce a mapper based on our *decoupled off-chip/on-chip approach* to accelerate mapping space exploration. Our mapper decomposes the original higher-dimensional mapping space of operators into two lower-dimensional off-chip and on-chip subspaces, and then optimizes the off-chip subspace followed by the on-chip subspace. This decomposition's motivation is to dramatically reduce the search space's size and prioritize the optimization of off-chip data movement, which requires significantly more energy and latency than the on-chip data movement [8]. In contrast to prior works [12, 34, 54] that use a single cost model for mapping space exploration, we use different approaches and different cost models for these subspaces, such as a classical **distinct-block (DB)** locality cost model [16, 44] to explore the off-chip subspace, and the MDC's cost model [24] for the on-chip subspace. We used a different cost model for the off-chip subspace exploration because the MDC's cost model requires the full specification of a mapping and does not work with a partial specification (e.g., off-chip mapping part of a full mapping). Even though we restrict our attention to mapping an operator onto a single accelerator, our decoupled approach can be extended to multiple accelerators within a chip and then to a distributed setup by prioritizing the optimization based on the data movement costs.

We implemented our overall approach in a tool called *Marvel*, and a benefit of our approach is that it applies to any operator conformable with the MDC notation. Marvel can be leveraged for both training and inference, as long as the required operators are conformable with MDC notation. Given a conformable DNN operator, workload sizes, and a target accelerator configuration, Marvel explores the mapping space of the operator using the decoupled approach and then outputs the mappings optimized for runtime and energy. Overall, our approach reduced the mapping space by an $O(10^{10})$ factor for the convolution operators in four major CNN models (AlexNet, VGG16, ResNet50, and MobileNetV2) while generating mappings that demonstrate a geometric mean performance improvement of 10.25× higher throughput and 2.01× lower energy consumption compared with three state-of-the-art mapping styles from past work. We also evaluated our approach over the other operators (GEMM, LSTM, and MLP) and compared them with past work optimizers.

2 BACKGROUND

In this section, we provide a brief overview of spatial accelerators and also the MDC notation to describe computation and mappings onto the spatial accelerators.

2.1 Spatial Accelerators

Spatial DNN accelerators based on ASICs and FPGAs have emerged to address extreme demands on performance and energy efficiency of CNN layers [5, 7, 20, 33, 35, 46]. Such accelerators are built using an array of PEs to provide high parallelism and use direct communication instead of via shared memory for energy efficiency. An abstract model of spatial accelerators is shown in Figure 2, where each PE of an accelerator consists of a single/multiple ALU(s) dedicated for **multiply-accumulate operations (MACs)** and a local scratchpad (L1 buffer). In addition, accelerators employ various NoCs for direct communication among PEs and between the PE array and the L2 scratchpad buffer. The interconnection network often supports multi-casting data to multiple PEs, which can reduce the total number of data reads from the L2 buffer to PEs. Unlike GPU cores, PEs can communicate with adjacent PEs (data forwarding) using a NoC, which can significantly reduce the number of L2 buffer accesses with high energy overhead. Accelerators also typically employ a

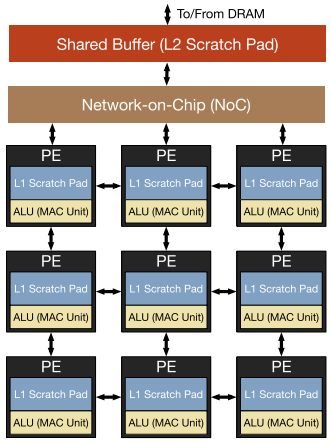

Fig. 2. The abstract spatial accelerator model, which is pervasive in many state-of-the-art accelerators [7, 20, 33].

large shared L2 scratchpad buffer to stage data from DRAM and also partial accumulations from PE arrays. Both L1 and L2 scratchpad buffers are software-controlled memories—for instance, the programmer/compiler directly controls contents of the buffer, unlike cache memories, which implicitly manage them, and this is because the memory traffic in accelerators is known in advance. Many spatial accelerators can be further interconnected together to create a scale-out system [11].

Systolic arrays [20, 53] belong to spatial accelerators and entirely rely on the point-to-point connection among adjacent PEs for input data distribution and partial sum accumulations. Although systolic arrays can provide high throughput and energy efficiency, they lack flexibility in its dataflow due to their rigid NoC architecture. Such inflexibility allows limited dataflow styles, which can lead to low compute unit utilization depending on the operator and its dimensions.

2.2 MDC Notation

The MDC notation, for expressing a DNN operator onto a spatial accelerator, consists of two aspects: (1) operator computation and tensor sizes, and (2) data mapping directives over tensor dimensions. A major novelty of the MDC notation is that the data mappings of tensors across space (PEs) and time are explicitly specified using a set of data mapping directives. This explicit specification enables the underlying cost model to estimate reuse behavior of a mapping precisely and also faster. We briefly describe the data mapping directives using a sample mapping (shown in in Figure 3(b)) for the CONV1D operator onto an accelerator having two PEs.

First, *TemporalMap (size, offset) d* specifies a distribution of the tensor dimension *d* index values across timesteps in a PE, and the mapped set of dimension indices is same across PEs. The `size` parameter refers to the number of contiguous indices mapped, and the `offset` parameter describes the shift in the starting indices of *d* across consecutive timesteps. For instance, the directive `TemporalMap(2,2)` d_w in the running example represents the distribution of first dimension (d_w) indices of the weight tensor, with two indices mapped in each timestep (e.g., d_w={0,1} in PE0 and PE1 at t = 0). In addition, the offset of two denotes the increment in the d_w index after each timestep (e.g., d_w={0,1} at t = 0 to d_w={2,3} at t = 1) until the extent of the d_w dimension is explored.

Second, *SpatialMap (size, offset) d* specifies a distribution of the tensor dimension *d* index values across PEs. The `size` parameter refers to the number of contiguous indices mapped, and the `offset` describes the shift in the starting indices of *d* across adjacent PEs. For instance, the

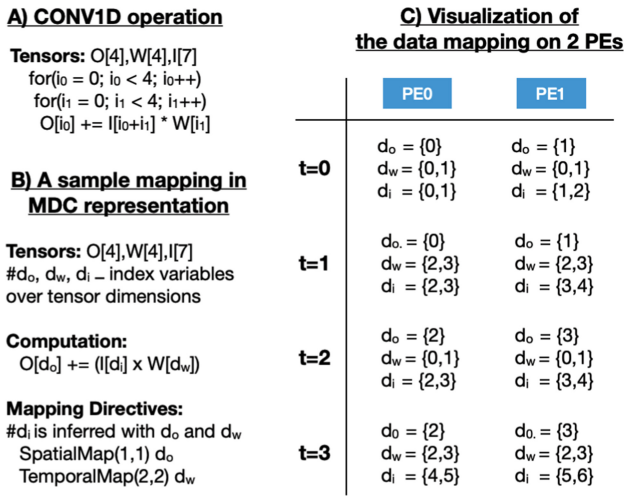

Fig. 3. A CONV1D's mapping in the MDC notation along with a visualization of its data mappings.

directive `SpatialMap(1,1)` d_O in the running example represents the distribution of first dimension (d_O) indices of the output tensor, with one index mapped to each PE in a given timestep (e.g., d_O={0} in PE0 and d_O={1} in PE1 at t = 0). If the number of PEs is not sufficient to cover all indices of the dimension mapped, then the mapping is folded over time across the same set of PEs.

Third, we have *directive order*, in which the sequence of spatial and temporal map directives in a mapping dictates the change of PE data mappings across time. Similar to a loop order, all the dimension index values corresponding to a mapping directive must be explored before its immediate outer mapping directive exploring its next set of indices. For instance, the sequence of directives in the running example (i.e., spatial map over d_O followed by temporal map over d_W) dictates that all the dimension index values of the weight tensors must be explored before exploring the next set of d_O indices. The order in this example results in accumulating partial results of the output before computing another output, popularly referred to as "output stationary" mapping [13]. However, the sequence notation has a limitation that it cannot capture scenarios where more than one dimension index value simultaneously changes over time (except at the dimension boundaries).

Fourth, the *clusters (size)* directive logically groups multiple PEs or sub-clusters, and the `size` parameter denotes the group size. For example, the `Cluster(2)` directive on an accelerator with four PEs arranges the PEs into two clusters with the cluster size as two, as shown in Figure 4(a). All the mapping directives above a cluster directive operate over the introduced logical sub-clusters (viewing each sub-cluster as a PE), whereas those below the cluster directive operate within a logical sub-cluster. The cluster directive is extremely useful in exploiting spatial distribution of more than one tensor dimension index value (e.g., row-stationary mapping [7]). In Figure 4(a), the tensor dimension d_w via `SpatialMap(2,2)` is spatially partitioned across the clusters where as the d_o via `SpatialMap(1,1)` is partitioned across the sub-clusters within a cluster.

Data blocking/tiling. Blocking is an essential mechanism to fit the required tensor's data across multiple levels of memory hierarchy of an accelerator and also exploit locality for better performance and energy efficiency. The clusters directive of MDC representation helps in realizing the data movement behavior arising from the blocking mechanism. Consider the example in Figure 4(b), where the `Cluster(2)` directive groups both the two PEs into a single cluster, and the sequence and sizes of temporal directives above the cluster dictate the block sizes and also the inter-block order.

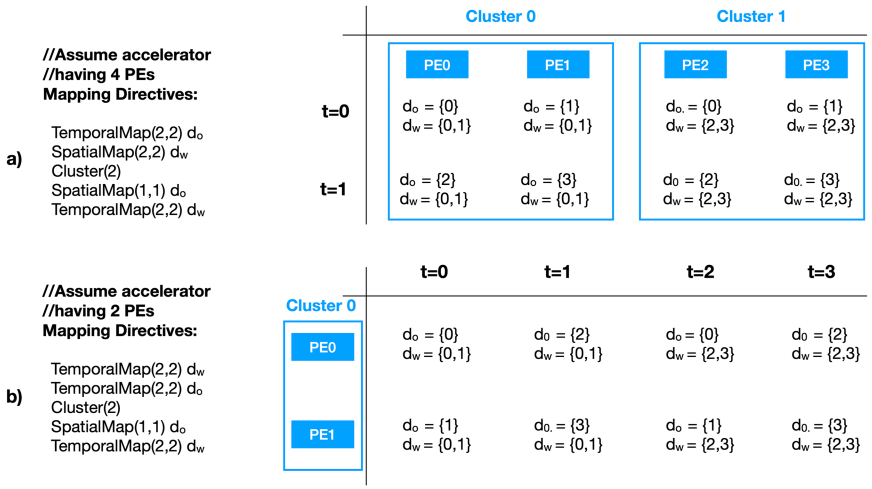

Fig. 4. Additional mappings built on the example in Figure 3 for describing (a) parallelism across multiple dimensions and (b) tiling behavior.

As can be observed from the usage of the preceding four directives, the PE data mappings are explicit in the notation, which enables the underlying cost model to estimate reuse behavior of a mapping precisely and also faster [24]. In addition, these directives capture a wide range of mappings/dataflow styles, including sophisticated mapping styles such as row-stationary in Eyeriss [7], weight-stationary in NVDLA [33], and output-stationary in ShiDianNao [13] accelerators. The GitHub repository [23] includes different mapping styles applied on several accelerators and also different DL models using the MDC notation. In addition, an interesting property of the MDC notation is that any mapping expressed in the notation is precisely analyzable using the MDC's cost model. However, finding whether a computation and any of its mappings are expressible in the MDC notation is still an open question, and we address this in the next section.

3 CONFORMABILITY RULES

In this section, we introduce a *set of conformability rules* to characterize the set of input operators and their mappings that can be expressed using the MDC notation. We discuss the rules over abstract loop nest description of operators that only describe the computation without any transformations for reuse and parallelization (e.g., CONV1D in Figure 5), and these rules are formed based on the computation and mapping directives of the MDC representation.

Rule 1: A conformable operator in its abstract loop nest description should be perfectly nested loops without any conditional statements inside the loop body.

The MDC notation restricts its computation to be uniform across all PEs at all timesteps. This restriction is satisfied only if a perfect loop nest encloses such uniform computation without any conditional statements. Most of the primitive DNN operators such as CONV2D, GEMM, and MLP (more in Table 1) can be expressed in the form of perfectly nested loops without any conditionals. However, there can be the implementation of certain operators, such as the fusion of two convolutions, where each PE requires executing the non-uniform computation and describing such mappings are not possible with the existing MDC notation. In addition, existing popular cost models for templated spatial architecture such as Timeloop [34] do not support such mappings.

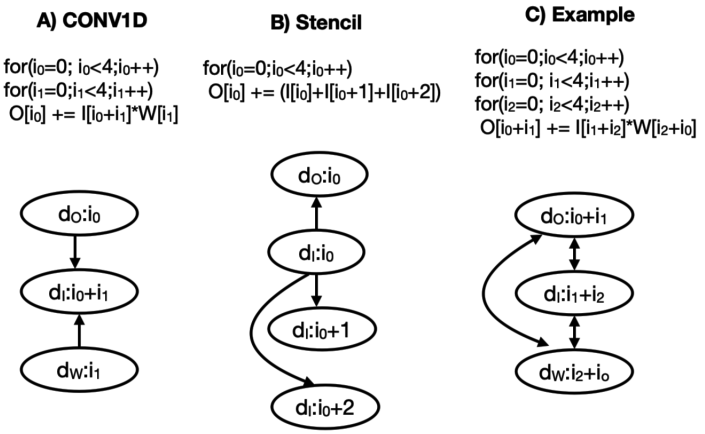

Fig. 5. The DDG of simple operators such as CONV1D and stencil satisfying rule R3 and an example violating rule R3. $d_O/d_I/d_W$: tensor dimension index variables corresponding to the output, input, and weight tensors.

Rule 2: The loop nest must not have any dependences (flow-, anti-, output-) except reduction dependences, and thus the loops can be freely reordered.

The MDC notation restricts the input and output tensors of an operator/computation to be different, resulting in not having any flow- and anti-dependences between the tensors. However, the notation can support reductions (e.g., add, max, min) in the computation, leading to supporting reduction dependences, such as flow-, anti-, output-dependences only on the output tensor. Like rule 1, most primitive DNN operators have only reduction dependences, except few operators such as parametric multi-step LSTMs with flow dependences. These multi-step LSTMs are unrolled in practice, leading to a sequence of loop nests, where each loop nest satisfies all the conformability rules, thereby enabling the MDC notation to capture each nest as a separate operator.

Rule 3: The dimension dependence graph of the perfectly nested loop must have a topological ordering, and the subscripts of each dependent dimension index variable of the DDG should be expressible as affine functions of the loop iterators.

The directive order (i.e., the sequence of mapping directives) of the MDC notation dictates the data mapping changes to PEs across time. As described in Section 2.2, the directive order has limitations in capturing more than one tensor dimension index variable changing simultaneously over time (except at boundaries). We introduce a new directed graph called the **dimension dependence graph (DDG)** to find possibilities of such data movement behaviors in an input computation/operator.

Nodes of the DDG. Each node of a DDG denotes a tensor dimension index variable along with a subscript referenced in that dimension. For instance, the node ($d_I : i_0+i_1$) in Figure 5(a) represents the subscript i_0+i_1 used in the input tensor (I) with dimension name d_I. All such unique tensor dimension index variables with their subscripts are part of the DDG as nodes.

Edges of the DDG. The edges of the DDG denote deduction relations (i.e., a destination node's subscript) can be completely constructed from the source node's subscripts of its incoming edges. These deduction relations are constructed as follows:

(1) An edge is added from a node having a **single index variable (SIV)/multiple index variable (MIV)** subscript[2] to a node having a MIV subscript if there is a common loop iterator

[2]The SIV subscript involves one loop iterator, whereas MIV subscripts involve more than one loop iterator [2].

in their subscripts. For example, there is a directed edge from the node ($d_O : i_0$) to ($d_I : i_0+i_1$) in Figure 5(a) because they have a loop iterator i_0 in common.

(2) All the SIV subscripts are grouped based on their loop iterators, and then edges are added from the SIV subscript of a group having the lowest constant value (randomly choose if there exists multiple) to other SIV subscripts in the same group. For example, there is a directed edge from the node ($d_I : i_0$) to all the nodes ($d_I : i_0 + 1$), ($d_I : i_0 + 2$), and ($d_O : i_0$) in Figure 5(b).

(3) If the loop bound of an iterator (say `i`) is dependent on other loop iterators (say `j`), then construct an edge from a node with subscript having the iterator `i` to nodes having the iterator `j` in their subscripts. This is not common in DNN primitive operators because the loop bounds in these operators are generally fixed and do not vary with outer loop iterators.

Now, finding multiple dimension index variables changing simultaneously is reduced to the problem of finding a topological ordering in the DDG. In essence, the absence of a topological ordering indicates the presence of mutually dependent dimension index variables (e.g., the example in Figure 5(c)). In the presence of a topological ordering, the MDC notation requires only the data mappings of independent dimension index variables to be specified, and these variables are identified from the nodes of the DDG having zero in-degree. In the case of CONV1D in Figure 5(a), only the data mappings of dimension index variables related to output and weight tensors must be specified. The underlying MDC's cost model infers the dimension variable related to the input tensor, and such dimension variables must not be specified. Furthermore, the subscripts of dependent dimension index variables should be affine expressions of loop iterators to be analyzable by the MDC's cost model.

Rule 4: After loop nest normalization, the subscripts associated with each independent dimension index variable of a DDG should be simply loop iterators with positive unit coefficients and no constants.

A mapping directive (either spatial or temporal) over a dimension index variable restricts the index values to start from zero and increases with unit stride. These restrictions do not allow the index variable to have strided increments or negative strides. To characterize the preceding restrictions implication, we assume the abstract loop nest form of an input computation to be normalized—its loop iterators start from zero and have unit strides (i.e., the loops themselves need to be of the form (`for i = 0; i < X; i++`)).

As described earlier, the MDC notation requires specification of directives over only independent dimension index variables of the DDG. Hence, to support the restrictions, each subscript (in the normalized form) associated with an independent dimension index variable must be simply loop iterators with positive unit coefficients and no constants (e.g., i_0 for d_O, i_1 for d_W in Figure 5(a)). Despite forcing only unit strides in the loop itself, the dilation and stride parameters of convolutions operators can be captured by this notation as these parameters are affine and part of the dependent dimension index variable subscripts (thanks to affine expressions from rule 3).

Conformability. Finally, an input computation/operator is said to MDC conformable if the computation in its abstract loop nest form satisfies all the preceding four rules. This also means that any legal mapping of the computation on a spatial accelerator can be expressed using the MDC notation. Table 1 lists the set of popular DNN primitive operators and the conformability of these operators with the MDC notation. Both the training and inference variants of the operators are conformable. As can be seen, the MDC notation captures most of the DNN operators except parametric LSTMs, and the mappings of these operators can be analyzable by the MDC's cost model.

Table 1. Conformability of the Popular DNN Operators onto the MDC Notation (Y/N Refers to YES/NO)

DNN Operator	Types	Rule 1	Rule 2	Rule 3	Rule 4	Conformable to MDC
CONV1D	Regular	Y	Y	Y	Y	Y
CONV2D	Regular	Y	Y	Y	Y	Y
	Point-wise, Depth-wise	Y	Y	Y	Y	Y
	Strided, Dilated	Y	Y	Y	Y	Y
MLP	Fully connected	Y	Y	Y	Y	Y
Pooling	Max, Avg	Y	Y	Y	Y	Y
GEMM	Regular	Y	Y	Y	Y	Y
	Triangular	Y	Y	Y	Y	Y
LSTM	Single cell	Y	Y	Y	Y	Y
	Parametric multi-cell	Y	N	Y	Y	N
Element wise	Residual	Y	Y	Y	Y	Y
	ReLU	Y	Y	Y	Y	Y
Stencils	Regular	Y	Y	Y	Y	Y

4 TRANSFORMING A LOOP NEST MAPPING INTO MDC NOTATION

The MDC notation is powerful in expressing and reasoning complex mappings of DNN operators onto the diverse spatial accelerators, but explicitly writing and exploring such mappings can be error-prone and tedious. In addition, computer architects [34] and DNN compiler frameworks [6] view the computations and their mappings majorly in the loop nest form [29, 34, 56]. This section introduces an *automatic transformation* to translate a loop nest mapping of a conformable operator into its equivalent MDC notation. In this work, we assume the target spatial accelerator having three levels of the memory hierarchy (private L1 buffer, shared L2 buffer, and DRAM). However, our transformation can be easily extendable to more levels of hierarchy.

As described in Section 2.2, the MDC notation consists of two components: (1) computation and tensor sizes, and (2) data mapping directives over independent tensor dimensions. The statements enclosed in the perfectly nested loop form of an input conformable operator are used as the computation, and the tensor sizes are extracted from the workload configuration. The computation and tensor sizes of the MDC notation remain the same for all mappings of the operator and its workload configuration. Then, the DDG is constructed to identify the set of independent tensor dimension index variables. If there are no such independent dimension index variables, then the operator is discarded as non-conformable. Otherwise, we will translate the mapping in loop nest form into data directives of the MDC notation.

4.1 Generation of Data Mapping Directives

From rules 1 and 2 in Section 3, the loops of a conformable operator can be freely reordered, and hence it is legal to apply *multi-level tiling* to exploit temporal reuse across each level of the memory hierarchy and also to exploit parallelism via PEs of the accelerator. Each behavior arising from tiling, reuse, and parallelizing an operator onto a spatial accelerator is referred to as a "mapping." The different aspects of mapping are briefly described in the following with a running example of CONV1D's loop nest mapping (shown in Figure 6(c)) over a 3-level accelerator.

Multi-level tiling tile sizes. A mapping includes tile sizes of all loop iterators for each level of tiling: (1) level-1 tiling for the private L1 scratchpad buffer inside a PE, (2) level-2 tiling for the parallelism across PE array, and (3) level-3 tiling for the global L2 scratchpad buffer shared by all PEs.

Inter-tile loop orders. A mapping also includes inter-tile loop orders[3] to describe the execution order of tiles exploiting various reuse opportunities. For instance, the level-2 inter-tile loop order (i.e., the execution order of level-2 tiles) exploits spatio-temporal reuse via the on-chip interconnect. In addition, level-3 inter-tile loop order exploits temporal reuse via the on-chip L2 buffer. However, the level-1 inter-tile loop order does not reflect any reuse because these loops essentially reflect parallelism. The point loop's loop order does not provide any reuse opportunities because there is no more intermediate staging between the PE and its private L1 buffer.

An n-level tiling over an input conformable operator will have `n` set of tile loops (including parallel loops) and a set of point loops. Each set of loops can have a different data movement (reuse) behavior based on its sizes and loop order. We introduce a term called *region* to denote a sequence of data mapping directives (e.g., region R1 in Figure 6(d)) without any cluster directives. Each region captures the data movement behavior present in each set of tile loops. Rules 3 and 4 help in constructing a one-to-one mapping between the tile loop orders and directive order—for example, if rule 3 (topological) is violated, we cannot find MDC's directive sequence order to reflect the required data movement behavior. Given a mapping in multi-level tile sizes and inter-tile loop orders, our approach transforms the mapping into the MDC notation as described next.

Point loops. As described in rule 4, each subscript associated with an independent dimension variable is simply a unique loop iterator. Our approach translates each loop of point loops into a temporal map directive over the corresponding independent tensor dimension index variable with `size` and `offset` parameters of the directive the point loop size. For example, the point loop t_{1i} with tile size T_{1i} in Figure 6(c) is directly translated into `TemporalMap(`T_{1i}`,`T_{1i}`)` d_O in the region R1 shown in Figure 6(d). Since the loop order among the point loops does not provide any reuse benefits, the directive order in region R1 does not matter. Lines 2 and 3 in Algorithm 1 are related to building a sequence of temporal maps in region R1 corresponding to the point loops in the given mapping.

Parallel loops. Since each independent dimension index variable is uniquely associated with a loop iterator, parallel execution of each loop iterator introduces a different data movement behavior.

Hence, we introduce a region with a spatial map over the dimension index variable associated with the parallel loop and the temporal maps for the rest of the dimension variables in that region for each parallel loop. For example, there are two regions with the names R2 and R3 for the parallel loops corresponding to t_{2j} and t_{2i}, respectively. In addition, the dimension d_W associated with the iterator t_{2j} and the dimension d_O associated with the iterator t_{2i} are translated into spatial maps in R2 and R3 regions, respectively. The `size` and `offsets` of each spatial map over a dimension variable is derived from the strides of the parallel loop iterators corresponding to the dimension variable. The order of directives in each region corresponding to parallel loops does not matter because the number of iterations arising from the rest of the temporal maps is one. Each region corresponding to a parallel loop (except the innermost) is ended with a cluster directive with `size` as the number of iterations in the parallel loop (lines 5–13 in Algorithm 1). For example, the region R3 is ended with a cluster directive with `size` as the number of iterations of the loop t_{2i}.

Inter-tile loops. For each set of tile loops excluding parallel loops, our transformation generates a region by creating a temporal map directive for each loop of the set with the `size` and `offset` of the directive as the loop stride. For example, the inter-tile loop t_{3j} with stride as T_{2j} in Figure 6(c) is directly translated into `TemporalMap(`T_{2j}`,`T_{2j}`)`d_I in the region R4 shown in Figure 6(d). The loop order governs the order of directives in a region among the corresponding tile loops. For example,

[3] An n-dimensional loop nest after one level of tiling will have 2n loops. The outer n-loops are referred to as inter-tile loops and the later n-loops as intra-tile loops. The innermost n-loops after multi-level tiling are called *point loops*.

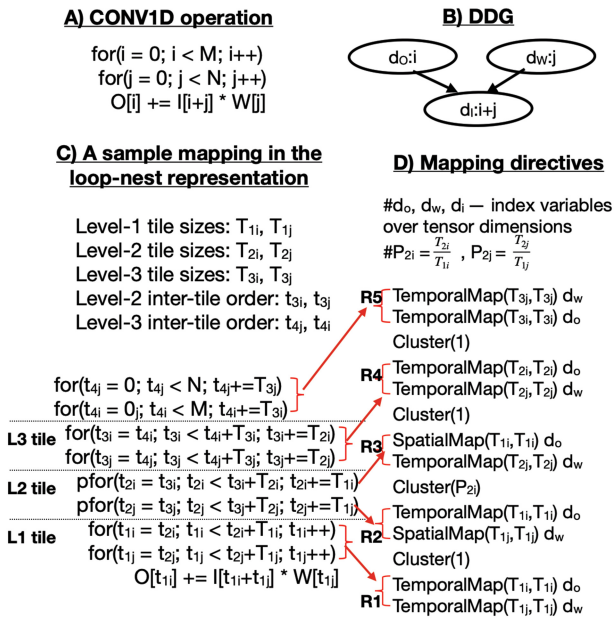

Fig. 6. A brief overview of the CONV1D's mapping expressed in the loop nest form and its translation into the MDC notation with data mapping directives.

ALGORITHM 1: Transforming a mapping in loop nest form into MDC mapping directives.

```
1  Level-0-directives, Level-1-directives, Level-2-directives, Level-3-directives = { }
2  for every loop in the operation do
3      Level-0-directives += TemporalMap(level-1-tile-sizes[loop], level-1-tile-sizes[loop])
4  cluster-size = 1, level-1-sizes = level-1-tile-sizes
5  for every parallel-loop in the operation do
6      Level-1-directives += Cluster(cluster-size)
7      for every loop in the operation do
8          if loop == parallel-loop then
9              Level-1-directives += SpatialMap(level-1-sizes[loop], level-1-sizes[loop])
10             level-1-sizes[loop] = level-2-tile-sizes[loop]
11         else
12             Level-1-directives += TemporalMap(level-1-sizes[loop], level-1-sizes[loop])
13     cluster-size = level-2-tile-sizes[loop]/level-1-tile-sizes[loop]
14 for every loop in the level-2-inter-tile-loop-order do
15     Level-2-directives += TemporalMap(level-2-tile-sizes[loop], level-2-tile-sizes[loop])
16 for every loop in the level-3-inter-tile-loop-order do
17     Level-3-directives += TemporalMap(level-3-tile-sizes[loop], level-3-tile-sizes[loop])
18 Directives = Level0-directives + Level1-directives + ∑_{i=1}^{2}(Cluster(1) + Level(i)-directives)
```

the level-3 inter-tile loop order (t_{3j}, t_{3i}) dictates the temporal map over d_W outer compared to the temporal map over d_O in region R5. Furthermore, each region is separated by a cluster directive to support different data movement behaviors across each set of tile loops (lines 14–18 in Algorithm 1).

5 MAPPING SPACE EXPLORATION

In addition to the different aspects of mapping described in Section 4, we also consider a limited form of data layout choices (i.e., dimension reordering [28]) for operator tensors on the DRAM because the data movement between DRAM and on-chip global scratchpad buffer happens at the granularity of DRAM block sizes. Overall, the mapping space of an operator is a Cartesian product of six dimensions that represent different aspects of a mapping—for instance, (1) level-1 tile sizes, (2) level-2 tile sizes (parallelism), (3) level-2 inter-tile loop orders, (4) level-3 tile sizes, (5) level-3 inter-tile loop orders, and (6) data layout of tensors. *The conformability rules enable to have such a structured mapping space of the conformable operators.* Searching for optimal mapping can be challenging because of the massive space of the operator's legal mappings on the accelerator configurations. For example, there are over 10^{19} valid mappings for the convolution operator on average for mapping ResNet50 and MobileNetV2 models on a representative DNN edge accelerator. This search challenge can get exacerbated with the evolution of new computations (e.g., depth-wise) and diverse hardware accelerator configurations (e.g., tree-based interconnect [26]).

Our approach toward mapping space exploration is motivated by the observation that the off-chip data movement between DRAM and on-chip global scratchpad requires significantly more energy and latency than the on-chip data movement [8]. *Hence, we propose an approach referred to as "decoupled off-chip/on-chip" that decomposes the original higher-dimensional mapping space into two lower-dimensional off-chip and on-chip subspaces, and then optimizes the off-chip subspace followed by the on-chip subspace, which is constructed with the optimal mappings from the off-chip subspace.* The off-chip subspace consists of three dimensions of the original mapping space—level-3 tile size, level-3 inter-tile loop order, and data layouts—because they influence the off-chip data movement. Similarly, the on-chip subspace consists of the remaining three dimensions of the original space—level-1 tile sizes, level-2 tile sizes, and level-2 inter-tile loop order—because they contribute to parallelization and on-chip data movement. In contrast to prior work [12, 34, 54], we use different approaches and cost models for these subspaces—that is, a classical DB locality cost model [16, 44] to explore the off-chip subspace, and the MDC's cost model [24] for the on-chip subspace. We used a different cost model for the off-chip subspace exploration because the MDC's cost model requires the full specification of a mapping and does not work with a partial specification (e.g., off-chip mapping part of a full mapping). The overall approach is implemented as a stand-alone tool (shown in Figure 7) that takes a conformable operator, workload sizes, and a target accelerator configuration, then explores the mapping space of the operator using the decoupled approach, and finally outputs the mappings optimized for runtime and energy.

5.1 Solving Off-Chip Mapping Subspace

The goal of finding an optimal mapping in the off-chip mapping subspace is to minimize off-chip data movement between DRAM and the L2 buffer of an accelerator. In our work, we assume the L2 buffer to be a software-managed scratchpad buffer, and *reducing the off-chip data movement*[4] *is equivalent to finding a level-3 tile that has highest arithmetic intensity* because the highest arithmetic intensity results in higher reuse and less data transfer.

In our approach, we consider the classical DB locality cost model [16] to measure the off-chip data movement cost, which was developed as part of the memory cost analysis to guide the automatic selection of loop transformations and also optimal tile size selections [44, 45, 47] in IBM XL compilers. The DB model is a good choice for our approach since it only focuses on optimizing for

[4]In case of non-software-managed scratchpad buffers, reducing data movement between DRAM and the L2 buffer is equivalent to finding a level-3 tile whose memory footprint can fit into the L2 buffer and is maximum.

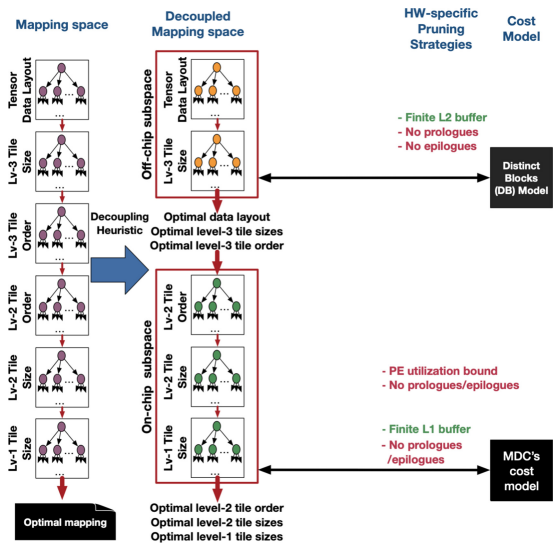

Fig. 7. An overview of our approach along with pruning strategies for searching mapping space of convolutions. The pruning strategies in green color preserve optimal mappings, whereas the strategies in red color may prune optimal.

off-chip data movement. Moreover, it is limited to only perfectly nested loops, and conformable operators are perfectly nested loops as per the rule R1 in Section 3.

The DB model starts with data layouts of multi-dimensional arrays and a parametric tiled version of a perfectly nested loop. The model symbolically estimates the off-chip data movement cost involved in a tile of computation by measuring the number of the distinct number of DRAM blocks required for all the references in the tile of computation. For instance, assume that an array `I` is laid out in the row-major order layout, then the distinct number of DRAM blocks (with DRAM block size as `B` and tile sizes T_X, T_Y) required for a reference `I[y][x+y]` enclosed in a triply nested loop with iterators `x, y, z` is computed as follows:

$$DB_I(T_X, T_Y) \approx \left(\left\lceil \frac{T_X + T_Y}{b} \right\rceil\right) \times (T_Y).$$

In the preceding formulation, the innermost access of the reference is divided by the block size,[5] because the data movement with DRAM happens in multiples of block sizes. The total data movement cost (DMC), a.k.a. memory cost per iteration, involved in a tile is computed as the number of distinct DRAM blocks required for all references in the tile divided by the total number of iterations in the tile. The optimal level-3 tile sizes and data layouts are computed by minimizing the data movement cost function for every layout and tile sizes in the off-chip mapping subspace with the two constraints: for instance, (1) the tile size of a loop should be greater than 0 and should not exceed its corresponding loop bound, and (2) the total data required (including double buffering) for a level-3 computation tile should fit into the on-chip L2 buffer. In the past, determining optimal tiles using the DB model was modeled as a geometric program, transformed then into a convex optimization problem [39, 40], and further solved using integer geometric programming frameworks instead of enumeration. Marvel currently has support for doing an exhaustive search

[5]Setting block size to one ignores the impact of data layouts that we consider in our approach.

(feasible because of only one level of tiling for off-chip data movement) and also using the integer geometric programming formulation for the tile size calculations.

After computing the optimal level-3 tile sizes and data layouts of tensors, our approach computes the partial derivatives (slopes) of the data movement cost function, based on the optimal data layout choice, with respect to parametric level-3 tile sizes, and evaluate the partial derivatives by substituting optimal level-3 tile sizes. The key insight is that having a higher negative value of a partial derivative along a loop indicates the lesser distinct number of elements referenced along the loop (i.e., highest reuse along the loop). It is suggested to keep it in the innermost position to exploit maximum temporal reuse, and similarly, the rest of the loops are ordered based on their partial derivative values.

Rationale for using the DB model. The DB model is a good choice for our approach, since the model only focuses on optimizing for off-chip data movement, and in addition, it focuses only on a perfectly nested loop, and conformable DNN operators are perfectly nested loops.

5.2 Solving On-Chip Mapping Subspace

The on-chip mapping subspace is constructed based on the optimal values of level-3 tile sizes. Our approach explores the constructed subspace to find optimal mappings for each of the three optimal goals: lower runtime (higher throughput), lower energy consumption, and lower energy delay product. For each mapping of the constructed subspace, our approach transforms the mapping in its loop nest form into its equivalent MDC notation (described in Section 4). Then, our approach uses the MDC's cost model [24] to estimate various metrics such as latency and energy of each mapping in the on-chip subspace. The MDC's cost model precisely computes performance and energy, accounting for under-utilization, edge conditions, and data reuse or movement across time (via L1/L2 buffers [7]), space (via broadcast links [26]), and space-time (via neighboring links [9, 20]) without requiring explicit RTL/cycle-level simulations or access to real hardware.

ALGORITHM 2: Our approach to explore on-chip mapping subspace.

```
1 for every level-2 inter-tile loop order do
2     for every level-2 tile size do
3         Hardware pruning: PE utilization bound
4         Hardware pruning: No prologues/epilogues
5         for every level-1 tile size do
6             Hardware pruning: Finite L1 size buffer
7             Hardware pruning: No prologue/epilogue
8             // Translate mapping into MDC form
9             Invoke the MDC's cost model → (runtime, energy, and other metrics)
```

Algorithm 2 shows an overview of our approach in exploring the on-chip mapping subspace along with pruning strategies. We introduce a parameter referred to as "PE utilization bound (p)" to prune search space of level-2 tile sizes by bounding the overall PE array utilization to be at least the parameter p. The preceding technique is beneficial in finding optimal on-chip mappings with the optimization goal being throughput because the highest throughput is typically obtained at higher PE utilization rates [10]. Our approach also includes a pruning strategy to choose level-1 and level-2 tile sizes such that they do not result in any prologues or epilogues (i.e., the tile sizes are factors of loop bounds). The preceding strategy is used in all the prior mapping space explorers [12, 29, 34, 54, 56]. This strategy helps simplify the design of a PE and control signal generation inside

Table 2. Operators, Occurrences, and MDC Conformability in DNN Models of MLPerf [38], VGG16, and AlexNet

DNN Operator	MLPerf Suite					VGG16	AlexNet	MDC Conformable?
	MobileNetV1	ResNet50	SSD-MobileNet	SSD-ResNet34	GNMT			
Regular CONV2D	15	54	34	51	0	16	9	✓
Depth-wise CONV2D	13	0	13	0	0	0	0	✓
Bias Add	1	1	12	12	0	1	1	✓
Batch Normalization	13	20	13	15	0	0	0	✓
ReLU	27	49	35	37	0	15	8	✓
Softmax	1	0	0	1	0	1	1	✓
Avg pooling	1	0	0	0	0	0	0	✓
Max Pooling	0	1	0	1	0	5	3	✓
GEMM	0	0	0	0	9	0	0	✓

	Accelerator platform (P1) (Eyeriss-like [7])	Accelerator platform (P2) (Edge/IoT-like)
#PEs	168	1024
Clock frequency	200 MHz	200 MHz
GigaOpsPerSec(GOPS)	67.2	409.6
NoC bandwidth (GB/s)	2.4	25.6
L1 buffer size	512B	512B
L2 buffer size	108KB	108KB

Fig. 8. Accelerator setups in our evaluation.

the accelerator. All of the pruning strategies mentioned previously can be enabled/disabled in Marvel by passing them as input parameters.

6 EXPERIMENTAL EVALUATION

In this section, we begin with coverage of MDC conformable operators in the existing popular DNN models. Then, we preset an overview of the experimental setup used in the evaluation of our decoupled off-chip/on-chip approach. We also present the evaluation of mappings generated by Marvel for a wide variety of DNN operators (CONV2D, GEMM, MLP, and LSTM) and discuss insights from the mappings while comparing them with previous work.

6.1 Coverage of MDC Conformable Operators

We have used the TensorFlow profiler to identify the operators in the DNN models of MLPerf suite (i.e., MobileNet V1, ResNet50, SSD-MobileNet, SSD-ResNet34, and GNMT). We also included VGG16 and AlexNet models in our evaluation. Table 2 lists those primitive operators and their occurrences in each DNN model. All the identified primitive operators are conformable with the MDC notation, and also we did not have to rewrite any of those operators to make it MDC conformable. Both the training and inference variants of those primitive operators are conformable.

6.2 Evaluation of Decoupled Off-Chip/On-Chip Approac

Target accelerators. Marvel is applicable to any spatial accelerator since it abstracts accelerator details as #PEs, L1/L2 buffer sizes, NoC bandwidth, reduction/multi-cast support, and so on, which can be used to model a wide variety of accelerators including Eyeriss [7], NVDLA [33], TPU, and xDNN.

Due to space limitations, we present our evaluation for only two accelerator platforms (shown in Figure 8): an accelerator (Eyeriss-like [7]) having 168 PEs and 2.4 GB/s NoC bandwidth, and another accelerator having 1,024 PEs and 25.6 GB/s. We inherit the L1 buffer, L2 buffer, and clock frequency for both platforms from Eyeriss [7] (i.e., 512B L1 buffer, 108-KB L2 buffer, and 200-MHz

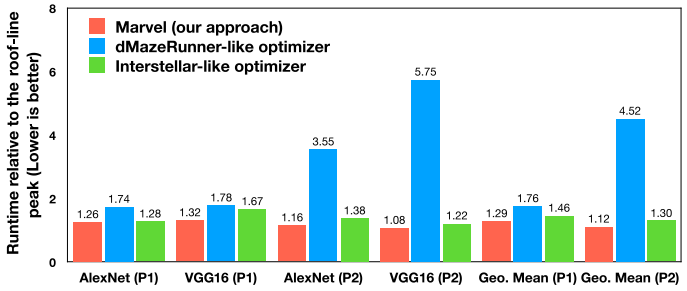

Fig. 9. Performance comparison of Marvel generated mappings with the mappings of the dMazeRunner-like optimizer [12] and Interstellar-like optimizer [54] relative to the roof-line peaks of the AlexNet and VGG16 models on both platforms (P1 and P2).

clock frequency). The bidirectional NoC used in our evaluation is a two-level hierarchical bus, which has support for multi-casting similar to Eyeriss.

Experimental variants. We have implemented few of the exploration strategies of recent optimizers such as Interstellar [54] and dMazeRunner [12] in our framework. For instance, the Interstellar optimizer focuses on parallelizing input and output channels of CONV2D operators, whereas the dMazeRunner optimizer focuses on parallelizing only output channels and a limited set of loop orders. We compare Marvel generated mappings for each workload and accelerator platform with three variants: (1) Marvel implemented Interstellar-like [54] optimizer generated mappings, (2) Marvel implemented dMazeRunner-like [12] optimizer generated mappings, and (3) roof-line peak based on the workload arithmetic intensities and accelerator configurations.

Methodology. We have evaluated all the mappings generated by the experimental variants using the MAESTRO cost model [24]. Moreover, the analytical cost model within the MAESTRO framework is validated against the RTL implementations of Eyeriss [7] and MAERI [26] on VGG16 and AlexNet models. We passed a pruning option to the Marvel to choose tile sizes that divide loop bounds evenly without any remainder, and this has been the consideration in the other approaches [12, 29, 34, 54, 56]. We also set the minimum PE array utilization bound as 0.1 (i.e., at-least 10% of the PE array should be mapped with computation). We apply 8-bit fixed point precision for all the tensors used in our evaluation.

6.2.1 Evaluation on CONV2D Variants. In our evaluation, we considered CONV2D (including depth-wise) operators of popular DNN models, such as AlexNet [22], VGG16 [48], ResNet50 [17], and MobileNetV2 [43]. We assumed a batch size of one because this assumption captures a low latency requirement use case and also represents a more challenging setup for energy efficiency and throughput [10], especially for edge-/mobile-class accelerators. In addition, these models encompass a broad spectrum of CONV2D operators such as regular, point-wise, depth-wise, strided variants with different filter shapes.

Comparison with the existing optimizers. Figure 9 presents the runtimes of mappings generated by Marvel, the dMazeRunner-like optimizer, and the Interstellar-like optimizer normalized to the roof-line peaks of the AlexNet and VGG16 models on both accelerator platforms (P1 and P2). Since each model has multiple CONV2D operations, we have added the runtimes of all the CONV2D operators in a model to present our evaluation at the level of DNN models. The Interstellar-like optimizer is almost equivalent to the brute-force exploration except that it allows exploiting parallelism along only input and output channels. As a result, the evaluation using the Interstellar-like optimizer is very time consuming (multiple days for MobileNetV2 and ResNet50), and hence we restricted the

Variants	Search space size		
	Min	Avg	Max
Original search space	2.7×10^{17}	9.4×10^{18}	1.8×10^{19}
Off-chip schedules search space after decoupling	7.3×10^{8}	3.6×10^{11}	1.3×10^{12}
On-chip schedules search space after decoupling	2.9×10^{7}	2.4×10^{10}	1.4×10^{11}
Off-chip schedules search space after decoupling + pruning	9.9×10^{5}	1.5×10^{8}	6.3×10^{8}
On-chip schedules search space after decoupling + pruning	3.8×10^{5}	5.9×10^{7}	2.4×10^{8}

Fig. 10. The statistics (min/avg/max) of the CONV2D mapping space in our evaluation and the resultant mapping subspaces after decoupling and pruning strategies.

comparison to only AlexNet and VGG16 models. Since we are comparing with the roof-line peak, we ignored comparing with TimeLoop [34]'s time-taking brute-force exploration to identify the best mappings.

As can be observed from Figure 9, Marvel generated mappings are geometrically 2.35× and 1.15× faster compared to the mappings obtained by the dMazeRunner-like optimizer and the Interstellar-like optimizer, respectively. The dMazeRunner-like optimizer focuses on exploiting parallelism along only output channels (in presence of unit batch size) to avoid inter-PE communication, and this results in under-utilization of the PE array for both models.

The Interstellar-like optimizer is able to perform close to Marvel, because the number of input and output channels in these models are larger (except at the initial layers). But, it can underperform for DNN models such as UNet [41], where input and output channels are smaller and output width and height are larger. Furthermore, our approach is able to identify mappings in seconds to few minutes for each operator on a local machine, unlike the Interstellar-like, dMazeRunner-like optimizer, which takes almost 1 to 5 hours for each operator. Figure 10 shows the impact of our decoupling and pruning strategies on the original search space of mappings of the four DNN models with an average reduction of $O(10^{10})$ in the mapping space.

Comparison with the popular mapping styles. The state-of-the-art mapping styles encoded in the hardware as dataflows are row-stationary from Eyeriss [7], weight-stationary from DLA [33], and output-stationary from ShiDianNao [13]. In our evaluation, we encoded the preceding mapping styles in the form of parallelization and loop order constraints on the on-chip mapping space in our decoupled approach. For instance, DLA-like mappings [33] exploit parallelization across input and output channels, Eyeriss-like mappings [7] exploit parallelism along output width and filter width, and ShiDianNao-like mappings [13] exploit along output width and height. To briefly explain the mappings generated by our approach and its difference with respect to the state-of-the-art mapping styles, we consider two convolutions—a regular CONV2D from VGG16 and a depth-wise CONV2D from MobileNetV2—whose details are shown in Figure 11.

Impact of level-3 tile sizes. The CONV2D operator in VGG16 layer 1 has higher output width and height (P, Q) compared to the output and input channels (K, C). However, the level-3 tile size corresponding to output height is shrinked to fit into the on-chip buffer with maximum temporal reuse. As a result, our approach exploited parallelism along output width (P) and filters (K) to utilize the PE array maximum. However, none of the state-of-the-art mapping styles and also dMazeRunner-like/Interstellar-like optimizers exploit parallelism along P and K dimensions in their mapping.

Impact of modern operators. The modern DNN models such as MobileNetV2 have introduced depth-wise CONV2D operators, and these operators reduce the total number of MAC operations by not performing reduction across input channels, thereby sacrificing arithmetic intensity. As a result,

Loop iterators of CONV2D	CONV1 in VGG16			Bottleneck6_3_2 in MobileNetV2		
	Regular CONV2D			Depth-wise separable		
	Loop sizes	Level-3 tile sizes	Degree of parallelism	Loop sizes	Level-3 tile sizes	Degree of parallelism
Batch(N)	1	1	1	1	1	1
Filters (K)	64	64	8	1	1	1
Input channels (C)	3	3	1	576	64	32
Output width (P)	222	111	37	5	5	5
Output height (Q)	222	6	3	5	5	5
Filter width (S)	3	3	1	3	3	1
Filter height (R)	3	3	1	3	3	1

Fig. 11. Two layers from VGG16 and MobileNetV2 for brief discussion on our approach generated mappings. Level-3 tile sizes and degree of parallelism are part of the mappings identified by our approach on platform P2.

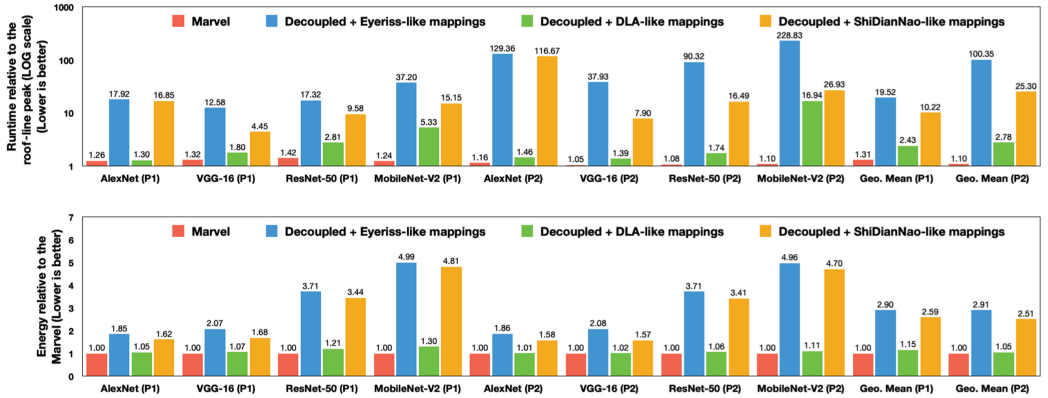

Fig. 12. Runtime and energy of Marvel generated mappings with popular mapping styles such as row-stationary from Eyeriss [7], weight-stationary from DLA [33], and output-stationary from ShiDianNao [13] for the AlexNet [22], VGG16 [48], ResNet-50 [17], MobileNet-V2 [43] models on platforms P1 and P2.

these operators have less parallelization opportunities and are often bounded by NoC bandwidth. For example, the depth-wise CONV2D operator in Figure 11 has the value of K set to one and also the level-3 tile size of C is shrinked to a smaller value to fit into the on-chip buffer with maximum temporal reuse. To fully leverage the PE array, our approach generated mapping exploited parallelism along three dimensions—input channels (C), output width (P), and output height (Q), where none of the prior state-of-the-art mapping styles and dMazeRunner-like/Interstellar-like optimizers exploited more than two levels of parallelism. Furthermore, the performance of the generated mapping was close to the roof-line peak, which is dominated by NoC bandwidth. The overall performance (runtime) and energy comparison of Marvel generated mappings with respect to the prior state-of-the-art mapping strategies is shown in Figure 12.

6.2.2 Evaluation on GEMM. We considered GEMM workloads from recent work [37]. An interesting aspect of these workloads is that they are irregular in their shapes, making the rigid accelerators (e.g., TPUs) hard to reach their peak utilization [37]. A summary of these workloads are shown in Figure 13, where M, N, and K refer to the number of rows, columns of first matrix followed by the columns of second matrix. As can be seen from Figure 14, the runtimes of Marvel generated mappings are only 1.24× and 1.10× higher relative to the roof-line peaks of accelerator

Workload	Application	Dimensions		
		M	N	K
GNMT	Machine Translation	128	2048	4096
		320	3072	4096
		1632	36548	1024
		2048	4096	32
DeepBench	General Workload	1024	16	500000
		35	8457	2560
Transformer	Language Understanding	31999	1024	84
		84	1024	84
NCF	Collaborative Filtering	2048	1	128
		256	256	2048

Fig. 13. GEMM workloads taken from [37].

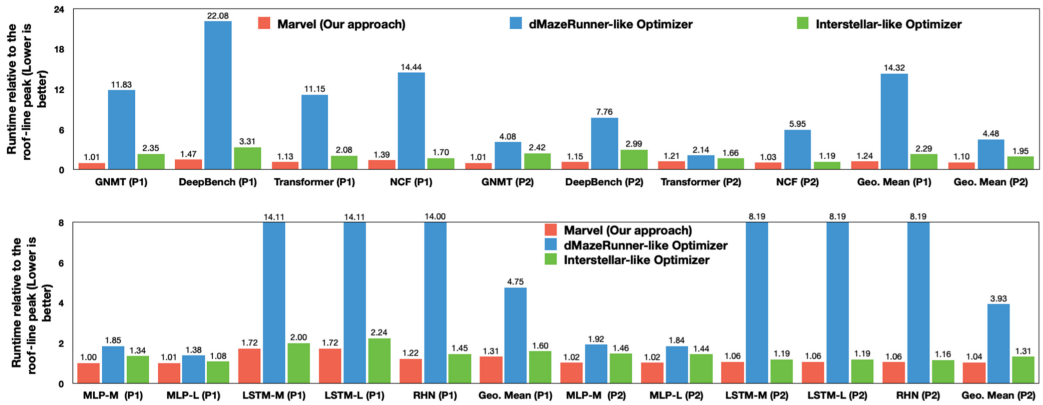

Fig. 14. Comparison of Marvel generated mappings with the mappings of dMazeRunner-like [12] and Interstellar-like optimizers [54] normalized to the roof-line peaks of the GEMM, LSTM, and MLP workloads.

Network	Layer	Input channels	Output channels
MLP-M	FC1	784	1000
	FC2	1000	500
	FC3	500	250
MLP-L	FC1	784	1500
	FC2	1500	1000
	FC3	1000	500

Network	Embedding size	Batch size
LSTM-M	500	128
LSTM-L	1000	128
RHN	1500	128

Fig. 15. MLP and LSTM workloads taken from Interstellar [54].

platforms P1 and P2, respectively, thereby demonstrating the closeness of mappings obtained using our approach to the peak. Furthermore, we observed that maximum reuse (spatial, temporal, spatio-temporal) is exploited only when all the dimensions of the GEMM operator are parallelized. Hence, Marvel generated mappings included parallelization of the three dimensions to make the PE array occupied along with exploiting maximum reuse. This is in contrast to other approaches—that is, the Interstellar-like optimizer focusing on the parallelizing only (N, K) dimensions and the dMazeRunner-like optimizer focusing on the parallelizing only (K) dimension.

6.2.3 Evaluation on MLP and LSTM. In this evaluation, we have considered the MLP and LSTM workloads from the work of Yang et al. [54], and a summary of these workloads is shown in Figure 15. Figure 14 presents the runtime of optimized mappings generated by Marvel, the dMazeRunner-like optimizer, and the Interstellar-like optimizer normalized to the roof-line peak of

each workload. Marvel generated mappings are 4.46× and 1.22× faster compared to the mappings obtained by the dMazeRunner-like optimizer and Interstellar-like optimizer for all the workloads on both accelerator platforms. The benefits of our approach compared to the dMazeRunner-like optimizer is higher because of its parallelization across only a single dimension (embedding size in case of LSTM and output channels in MLP) and also exploring only limited loop orders for reuse. Marvel is able to do better compared to the Interstellar-like optimizer by exploring more levels of parallelism to make the PE array occupied (e.g., only 1.04× higher relative to the roof-line peak on P2).

Summarizing, our decoupled off-chip/on-chip approach is applicable to any MDC conformable operator, and we have demonstrated our approach over multiple CONV2D variants, GEMM, MLP, and LSTM workloads that are popular in DNN models.

7 RELATED WORK

We categorize our discussion on prior work along the two directions: (1) expressiveness, formal boundary, and cost models, and (2) mapping exploration strategies.

7.1 Expressiveness, Formal Boundary, and Cost Models

A major difference between the mappers for spatial accelerators and CPUs/GPUs is the need for "accurate" cost models that can estimate close to real-world values for a given mapping of a kernel (operator) for finding efficient mappings [24, 34]. This is because the spatial accelerator's performance is sensitive to the mapping parameters—for example, a small change in the tile size or degree of parallelism would drastically change the latency or energy-efficiency numbers. Recently, multiple analytical cost models [12, 24, 34, 54] have been developed to estimate execution time and energy efficiency of mappings over spatial accelerators close to the real-world simulation/execution. These mappings are often expressed in the form of *loop nests*, and many cost models such as Timeloop [34], DMazeRunner [12], and Interstellar [54] are developed over the loop nest description of mappings. The loop nest syntax is very generic and can help accelerator architects or compilers express a wide range of mappings, *but the underlying cost models may not be able to estimate costs for all possible mappings that can be expressible in loop nests*. For example, a mapping for parametric multi-step LSTM operator may include loop skewing transformation to exploit pipelined (wave-front) parallelism [3], but it is not clear if the cost models mentioned previously can precisely estimate costs for such mappings.

Challenges with generic loop nest representation for cost models. The generic loop nest forms encompasses various control flow structures and data-access patterns that make it harder for the cost models to precisely reason. For instance, most of the cost models based on the loop nest form assumes the loops to have constant bounds, and this makes the analysis simpler by estimating various forms of reuse in a particular iteration and extrapolating with the total number of iterations in the loop. However, such analyses can break in presence of `if conditionals` in the loop body and also with variable loop bounds such as in non-rectilinear and sparse iteration spaces.

TVM compiler infrastructure [6] offers an ML-based cost model to find optimal implementations of convolutions on a variety of platforms including accelerators. However, we believe that such ML-based cost models are challenging for spatial accelerators for two reasons: (1) the statistical ML-based cost models are generally not accurate to precisely estimate performance and energy, and not accounting PE under-utilization, edge conditions can lead to significant imprecision, and (2) it requires training the ML-based cost models even for a slight change in number of PEs in the accelerator configuration, which makes it challenging to use for the design space exploration.

In this work, we have considered the recently introduced MDC notation [24] for expressing mappings onto generic templated spatial architectures because any mapping expressed in the notation

Table 3. Comparison of the MDC Notation with Prior Compilers in Terms of Expressiveness, Mapping Notation, and the Presence of Accurate Cost Models

Notation	Operator Expressiveness			Mapping Representation	Accurate Cost Models for Spatial Accelerators
	Loop Nest Structure	Array Subscripts	Iteration Domain		
MDC	Perfect	Affine	Affine	Data-centric	YES
TVM, TC, PlaidML, Stripe, ISAMIR	Perfect/Imperfect	Affine	Rectangular	Loop-centric	NO (YES for limited scenarios [54])
Polyhedral (e.g., Tiramisu)	Perfect/Imperfect	Affine	Affine	Loop-centric	NO
Generic loop nests (e.g., MLIR)	Perfect/Imperfect	Any	Any	Loop-centric	NO (YES only for limited scenarios [34])

is precisely analyzable using the MDC's cost model [25]. We introduced a set of conformability rules as a way to define a formal boundary over operators for precise and tractable cost analysis of the operator mappings using the MDC notation and its cost model.

Expressiveness. On the other side, even though high-level frameworks such as TVM [6], TC [50], PlaidML [36], Stripe [55], Polyhedral (Tiramisu [3]), and MLIR [27] have richer expressibility than the MDC notation, none of these frameworks yet have accurate cost models targeting flexible spatial accelerators. In addition, it is not clear if accurate cost models can exist for any operator expressed in their notations. An overview of the comparison of the MDC notation with prior notations in terms of expressiveness, mapping notation, and the presence of accurate cost models for spatial accelerators is shown in Table 3. Our approach, including mapping space exploration, can be plugged into popular frameworks such as TVM and MLIR for wide applicability.

In addition to Marvel supporting all the primitive operators supported by other frameworks (Figure 16), Marvel can also support operators having non-rectilinear iteration spaces (e.g., Symmetric GEMM [18]), which none of the other frameworks in Figure 16 support. Furthermore, a strong guarantee of our approach is that any operator conformable to the MDC notation can leverage our compiler along with the underlying accurate cost model. This is in contrast to other frameworks for spatial accelerators such as Timeloop and Interstellar, where there are no such guarantees.

MDC limitations. Currently, the MDC representation restricts the input computation to be a perfectly affine loops nest without any conditionals statements. Hence, our representation cannot support computations such as parametric LSTM operators and fusion of multiple convolution layers. However, such computations can be transformed into a sequence of MDC conformable operations by performing the transformations such as unrolling of parametric LSTM operations and loop distribution of imperfectly nested loops into perfectly nested loops. The polyhedral model is capable of covering the MDC limitations, but there does not exist any cost model based on the polyhedral model to accurately estimate performance aspects of a mapping on spatial accelerators, and this is a good line of research work for the future.

7.2 Mapping Exploration Strategies

Prior work [7, 57] focused on developing mappers specific to their architectures—for example, mRNA mapper [57] for the MAERI accelerator [26] and Auto-TVM [6] for the GEMM core of the VTA architecture [31] limiting their applicability to generic spatial accelerators. Prior work such as Zhang et al. [56] and Ma et al. [29] focused on spatial accelerators without L1 buffers inside a PE, again limiting their mapping space formulation. Furthermore, they do not employ accurate cost models and focus only on optimizing for runtime.

In addition, other prior works such as Interstellar and dMazeRunner fixed certain aspects of mapping space such as choice of parallel loops and loop orders, and these choices may not reflect the efficient mappings for a wide variety of DNN operators. To the best of our knowledge, Timeloop

Compiler/ Mapper	Target architecture	Target goal	Accurate cost models	Operators supported/ evaluated	Level-1 Tiling	Level-2 tiling			Level-3 tiling		Approach
					Tile sizes	Parallel loops	Degree of parallelism	Inter-tile order	Tile sizes	Inter-tile order	
mRNA	MAERI	Runtime, Energy	YES	CONV2D	NA	YES	YES	YES	NO	NO	Brute-force
Auto-TVM	VTA	Runtime	NO	CNNs	NA	YES	YES	YES	YES	YES	Annealing
Zhang et al.	Spatial	Runtime	NO	CONV2D	NA	FIXED	YES	FIXED	YES	YES	Brute-force
Ma et al.	Spatial	Runtime	NO	CONV2D	NA	FIXED	YES	FIXED	YES	YES	Brute-force
dMaze Runner	Spatial	Runtime, Energy	YES	CONV2D	YES	FIXED	YES	FIXED	YES	FIXED	Brute-force
Interstellar	Spatial	Runtime, Energy	YES	CONV2D, LSTM, MLP	YES	FIXED	YES	YES	YES	YES	Brute-force
Timeloop	Spatial	Runtime, Energy	YES	DeepBench, CNNs	YES	YES	YES	YES	YES	YES	Brute-force, random sampling
Marvel	Spatial	Runtime, Energy	YES	Any MDC Conformable	YES	YES	YES	YES	YES	YES	Decoupled

Fig. 16. Comparison of Marvel with prior mappers for spatial accelerators (mRNA [57], Zhang et al. [56], Ma et al. [29], Auto-TVM [6], dMazeRunner [12], Interstellar [54], and Timeloop [34]) for the mapping space exploration of operators. Our approach (Marvel) supports any operator conformable with the MDC notation.

is the only framework that considers all aspects of a mapping for a fully flexible spatial accelerator. However, it employs either an exhaustive linear search or a random sampling-based heuristic to explore the search space. In contrast to all preceding works, our approach considers all the aspects of mapping space and uses the decoupled strategy to navigate the mapping space efficiently.

Our work's key novelty is the formalization of MDC conformable operators using the four rules defined in Section 3, and with the conformability, our approach always generates a correct set of MDC directives corresponding to a loop nest mapping of the operator. The prior work introducing MDC directives [24] does not have any formalization and also any correctness checker over the usage of MDC directives. Further, the prior work is limited to hardware DSE and does not have any mapping explorer, unlike our approach.

8 CONCLUSION AND FUTURE WORK

In this article, we characterize the set of input operators and their mappings expressed in the MDC notation by introducing a set of conformability rules. The outcome of these rules is that any loop nest which is perfectly nested with affine tensor subscripts and without conditionals is conformable to the MDC notation, and a majority of the primitive operators in DL are such loop nests. In addition, our rules enable us to automatically translate a mapping expressed in the loop nest form to MDC notation and use the MDC's cost model to guide upstream mappers. Our conformability rules over the input operators result in a structured mapping space of the operators, and this enables us to introduce a mapper based on our decoupled off-chip/on-chip approach to accelerate mapping space exploration. We implemented our decoupled approach in a tool called *Marvel*,[6] and a major benefit of our approach is that it applies to any DNN operator conformable with the MDC notation. Our approach reduced the search space of CONV2D operators from four major DNN models from 9.4×10^{18} to $1.5 \times 10^8 + 5.9 \times 10^8 \approxeq 2.1 \times 10^8$, which is a reduction factor of 10 billion (Figure 10), while generating mappings that demonstrate a geometric mean performance improvement of 10.25× higher throughput and 2.01× lower energy consumption compared with three state-of-the-art mapping styles from past work. In the future, we envision (1) Marvel integration with the MLIR compiler infrastructure for wide usability, and (2) extending the MDC notation and its cost model to support non-conformable operators.

[6]Recently, we incorporated a variant of our decoupled off-chip/on-chip approach in the UNION framework [19].

ACKNOWLEDGMENTS

We thank Saurabh Malik, Natesh Raina, and Vaisakh Haridas for their insightful comments and help in the initial discussions of this work. We also thank the anonymous reviewers for their valuable feedback.

REFERENCES

[1] Google Cloud. 2019. Edge TPU: Google's Purpose-Built ASIC Designed to Run Inference at the Edge. Retrieved October 13, 2021 from https://cloud.google.com/edge-tpu/.

[2] Randy Allen and Ken Kennedy. 2001. *Optimizing Compilers for Modern Architectures: A Dependence-Based Approach*. Morgan Kaufmann.

[3] Riyadh Baghdadi, Jessica Ray, Malek Ben Romdhane, Emanuele Del Sozzo, Abdurrahman Akkas, Yunming Zhang, Patricia Suriana, Shoaib Kamil, and Saman Amarasinghe. 2019. Tiramisu: A polyhedral compiler for expressing fast and portable code. In *Proceedings of the 2019 IEEE/ACM International Symposium on Code Generation and Optimization (CGO'19)*. IEEE, Los Alamitos, CA, 193–205.

[4] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, et al. 2016. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316* (2016)

[5] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. 2014. Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'14)*.

[6] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Meghan Cowan, Haichen Shen, et al. 2018. TVM: An automated end-to-end optimizing compiler for deep learning. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation (OSDI'18)*. 579–594. http://dl.acm.org/citation.cfm?id=3291168.3291211.

[7] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. 2016. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. In *Proceedings of the International Symposium on Computer Architecture (ISCA'16)*.

[8] Yu-Hsin Chen, Tushar Krishna, Joel S. Emer, and Vivienne Sze. 2017. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal of Solid-State Circuits* 52, 1 (2017), 127–138.

[9] Yu-Hsin Chen, Tushar Krishna, Joel S. Emer, and Vivienne Sze. 2017. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal of Solid-State Circuits* 52, 1 (2017), 127–138.

[10] Yu-Hsin Chen, Tien-Ju Yang, Joel Emer, and Vivienne Sze. 2019. 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 (2019), 292–308.

[11] Eric S. Chung, Jeremy Fowers, Kalin Ovtcharov, Michael Papamichael, Adrian M. Caulfield, Todd Massengill, Ming Liu, et al. 2018. Serving DNNs in real time at datacenter scale with project brainwave. *IEEE Micro* 38, 2 (2018), 8–20. https://doi.org/10.1109/MM.2018.022071131

[12] Shail Dave, Youngbin Kim, Sasikanth Avancha, Kyoungwoo Lee, and Aviral Shrivastava. 2019. DMazeRunner: Executing perfectly nested loops on dataflow accelerators. *ACM Transactions on Embedded Computing Systems* 18, 5s (Oct. 2019), Article 70, 27 pages. https://doi.org/10.1145/3358198

[13] Zidong Du, Robert Fasthuber, Tianshi Chen, Paolo Ienne, Ling Li, Tao Luo, Xiaobing Feng, Yunji Chen, and Olivier Temam. 2015. ShiDianNao: Shifting vision processing closer to the sensor. In *Proceedings of the International Symposium on Computer Architecture (ISCA'15)*.

[14] Bruce M. Fleischer et al. bibinfoyear2018. A scalable multi-TeraOPS deep learning processor core for AI trainina and inference. In *Proceedings of the 2018 IEEE Symposium on VLSI Circuits*. IEEE, Los Alamitos, CA, 35–36. https://doi.org/10.1109/VLSIC.2018.8502276

[15] Clement Farabet, Camille Couprie, Laurent Najman, and Yann LeCun. 2013. Learning hierarchical features for scene labeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35, 8 (2013), 1915–1929.

[16] Jeanne Ferrante, Vivek Sarkar, and Wendy Thrash. 1991. On estimating and enhancing cache effectiveness. In *Proceedings of the International Workshop on Languages and Compilers for Parallel Computing*. 328–343.

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 770–778.

[18] Xu Shell Hu, Sergey Zagoruyko, and Nikos Komodakis. 2018. Exploring weight symmetry in deep neural networks. arXiv:1812.11027 [cs.LG]

[19] Geonhwa Jeong, Gokcen Kestor, Chatarasi Prasanth, Angshuman Parashar, Po-An Tsai, Sivasankaran Rajamanickam, Roberto Gioiosa, and Tushar Krishna. 2021. Union: A unified HW-SW Co-Design ecosystem in MLIR for evaluating tensor operations on spatial accelerators. In *Proceedings of the 30th International Conference on Parallel Architectures and Compilation Techniques (PACT'21)*.

[20] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, et al. 2017. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the International Symposium on Computer Architecture (ISCA'17)*. IEEE, Los Alamitos, CA, 1–12.

[21] Andrej Karpathy and Li Fei-Fei. 2015. Deep visual-semantic alignments for generating image descriptions. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR'15)*.

[22] A. Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet classification with deep convolutional neural networks. In *Proceedings of the NIPS Conference*.

[23] Hyoukjun Kwon. 2020. MAESTRO: An Open-Source Infrastructure for Modeling Dataflows Within Deep Learning Accelerators. Retrieved October 13, 2021 from https://github.com/maestro-project/maestro/tree/master/data/mapping.

[24] Hyoukjun Kwon, Prasanth Chatarasi, Michael Pellauer, Angshuman Parashar, Vivek Sarkar, and Tushar Krishna. 2019. Understanding reuse, performance, and hardware cost of DNN dataflow: A data-centric approach. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'52)*. ACM, New York, NY, 754–768. https://doi.org/10.1145/3352460.3358252

[25] Hyoukjun Kwon, Prasanth Chatarasi, Vivek Sarkar, Tushar Krishna, Michael Pellauer, and Angshuman Parashar. 2020. MAESTRO: A data-centric approach to understand reuse, performance, and hardware cost of DNN mappings. *IEEE Micro* 40, 3 (2020), 20–29. https://doi.org/10.1109/MM.2020.2985963

[26] Hyoukjun Kwon, Ananda Samajdar, and Tushar Krishna. 2018. MAERI: Enabling flexible dataflow mapping over DNN accelerators via reconfigurable interconnects. In *Proceedings of the 23rd International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'18)*. ACM, New York, NY, 461–475.

[27] Chris Lattner, Mehdi Amini, Uday Bondhugula, Albert Cohen, Andy Davis, Jacques Pienaar, River Riddle, Tatiana Shpeisman, Nicolas Vasilache, and Oleksandr Zinenko. 2020. MLIR: A compiler infrastructure for the end of Moore's law. arXiv:2002.11054 [cs.PL]

[28] Chao Li, Yi Yang, Min Feng, Srimat Chakradhar, and Huiyang Zhou. 2016. Optimizing memory efficiency for deep convolutional neural networks on GPUs. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis (SC'16)*. IEEE, Los Alamitos, CA, 633–644.

[29] Yufei Ma, Yu Cao, Sarma Vrudhula, and Jae-Sun Seo. 2017. Optimizing loop operation and dataflow in FPGA acceleration of deep convolutional neural networks. In *Proceedings of the International Symposium on Field-Programmable Gate Arrays (FPGA'17)*. ACM, Los Alamitos, CA, 45–54.

[30] Thierry Moreau, Tianqi Chen, and Luis Ceze. 2018. Leveraging the VTA-TVM hardware-software stack for FPGA acceleration of 8-Bit ResNet-18 inference. In *Proceedings of the 1st Conference on Reproducible Quality-Efficient Systems Tournament on Co-Designing Pareto-Efficient Deep Learning (ReQuEST'18)*. ACM, New York, NY, Article 5, 7 pages. https://doi.org/10.1145/3229762.3229766

[31] Thierry Moreau, Tianqi Chen, Luis Vega, Jared Roesch, Eddie Q. Yan, Lianmin Zheng, Josh Fromm, et al. 2019. A hardware-software blueprint for flexible deep learning specialization. *IEEE Micro* 39, 5 (2019), 8–16. https://doi.org/10.1109/MM.2019.2928962

[32] Mohammad Motamedi, Philipp Gysel, Venkatesh Akella, and Soheil Ghiasi. 2016. Design space exploration of FPGA-based deep convolutional neural networks. In *Proceedings of the 2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC'16)*. IEEE, Los Alamitos, CA, 575–580.

[33] NVIDIA. 2018. NVIDIA Deep Learning Accelerator (NVDLA). Retrieved October 13, 2021 from https://nvidia.org.

[34] Angshuman Parashar, Priyanka Raina, Yakun Sophia Shao, Yu-Hsin Chen, Victor A. Ying, Anurag Mukkara, Rangarajan Venkatesan, Brucek Khailany, Stephen W. Keckler, and Joel Emer. 2019. Timeloop: A systematic approach to DNN accelerator evaluation. In *Proceedings of the 2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS'19)*.

[35] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S. W. Keckler, and W. J. Dally. 2017. SCNN: An accelerator for compressed-sparse convolutional neural networks. In *Proceedings of the International Symposium on Computer Architecture (ISCA'17)*. 27–40.

[36] GitHub. 2020. PlaidML. Retrieved October 13, 2021 from https://github.com/plaidml/plaidml.

[37] Eric Qin, Ananda Samajdar, Hyoukjun Kwon, Vineet Nadella, Sudarshan Srinivasan, Dipankar Das, Bharat Kaul, and Tushar Krishna. 2020. SIGMA: A sparse and irregular GEMM accelerator with flexible interconnects for DNN training. In *Proceedings of the 2020 IEEE International Symposium on High Performance Computer Architecture (HPCA'20)*.

[38] Vijay Janapa Reddi, Christine Cheng, David Kanter, Peter Mattson, Guenther Schmuelling, Carole-Jean Wu, Brian Anderson, et al. 2019. MLPerf inference benchmark. arXiv:1911.02549 [cs.LG]

[39] Lakshminarayanan Renganarayana and Sanjay Rajopadhye. 2004. A geometric programming framework for optimal multi-level tiling. In *Proceedings of the 2004 ACM/IEEE Conference on Supercomputing (SC'04)*. IEEE, Los Alamitos, CA, 18. https://doi.org/10.1109/SC.2004.3

[40] Lakshminarayanan Renganarayana and Sanjay Rajopadhye. 2008. Positivity, posynomials and tile size selection. In *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing (SC'08)*. IEEE, Los Alamitos, CA, Article 55, 12 pages. http://dl.acm.org/citation.cfm?id=1413370.1413426.

[41] O. Ronneberger, P. Fischer, and T. Brox. 2015. U-Net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention*. Lecture Notes in Computer Science, Vol. 9351. Springer, 234–241. http://lmb.informatik.uni-freiburg.de/Publications/2015/RFB15a.

[42] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, et al. 2015. ImageNet large scale visual recognition challenge. *International Journal of Computer Vision* 115, 3 (2015), 211–252.

[43] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. MobileNetV2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 4510–4520.

[44] V. Sarkar. 1997. Automatic selection of high-order transformations in the IBM XL FORTRAN compilers. *IBM Journal of Research and Development* 41, 3 (May 1997), 233–264. https://doi.org/10.1147/rd.413.0233

[45] V. Sarkar and N. Megiddo. 2000. An analytical model for loop tiling and its solution. In *Proceedings of the 2000 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS'00)*. IEEE, Los Alamitos, CA, 146–153. http://dl.acm.org/citation.cfm?id=1153923.1154542.

[46] Hardik Sharma, Jongse Park, Divya Mahajan, Emmanuel Amaro, Joon Kyung Kim, Chenkai Shao, Asit Mishra, and Hadi Esmaeilzadeh. 2016. From high-level deep neural models to FPGAs. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture (MICRO'16)*.

[47] Jun Shirako, Kamal Sharma, Naznin Fauzia, Louis-Noël Pouchet, J. Ramanujam, P. Sadayappan, and Vivek Sarkar. 2012. Analytical bounds for optimal tile size selection. In *Proceedings of the 21st International Conference on Compiler Construction (CC'12)*. 101–121. https://doi.org/10.1007/978-3-642-28652-0_6

[48] Karen Simonyan and Andrew Zisserman. 2015. Very deep convolutional networks for large-scale image recognition. In *Proceedings of the International Conference on Learning Representations (ICLR'15)*.

[49] Alexander Toshev and Christian Szegedy. 2014. DeepPose: Human pose estimation via deep neural networks. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR'14)*.

[50] Nicolas Vasilache, Oleksandr Zinenko, Theodoros Theodoridis, Priya Goyal, Zachary DeVito, William S. Moses, Sven Verdoolaege, Andrew Adams, and Albert Cohen. 2018. Tensor comprehensions: Framework-agnostic high-performance machine learning abstractions. *arXiv preprint arXiv:1802.04730* (2018)

[51] S. Venkataramani, J. Choi, V. Srinivasan, W. Wang, J. Zhang, M. Schaal, M. J. Serrano, et al. 2019. DeepTools: Compiler and execution runtime extensions for RaPiD AI accelerator. *IEEE Micro* 39, 5 (2019), 102–111.

[52] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, et al. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144* (2016)

[53] xDNN n.d. Accelerating DNNs with Xilinx Alveo Accelerator Cards. Retrieved October 13, 2021 from https://www.xilinx.com/support/documentation/white_papers/wp504-accel-dnns.pdf.

[54] Xuan Yang, Mingyu Gao, Qiaoyi Liu, Jeff Setter, Jing Pu, Ankita Nayak, Steven Bell, et al. 2020. Interstellar: Using Halide's scheduling language to analyze DNN accelerators. In *Proceedings of the 25th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'20)*. ACM, New York, NY, 369–383. https://doi.org/10.1145/3373376.3378514

[55] Tim Zerrell and Jeremy Bruestle. 2019. Stripe: Tensor compilation via the nested polyhedral model. arXiv:1903.06498 [cs.DC]

[56] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong. 2015. Optimizing FPGA-based accelerator design for deep convolutional neural networks. In *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, New York, NY, 161–170.

[57] Zhongyuan Zhao, Hyoukjun Kwon, Sachit Kuhar, Weiguang Sheng, Zhigang Mao, and Tushar Krishna. 2019. mRNA: Enabling efficient mapping space exploration on a reconfigurable neural accelerator. In *Proceedings of 2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS'19)*. IEEE, Los Alamitos, CA.

Received April 2021; revised August 2021; accepted September 2021