Union: A Unified HW-SW Co-Design Ecosystem in MLIR for Evaluating Tensor Operations on Spatial Accelerators

Geonhwa Jeong*, Gokcen Kestor‡ §, Prasanth Chatarasi† §, Angshuman Parashar‖, Po-An Tsai‖, Sivasankaran Rajamanickam¶, Roberto Gioiosa‡ and Tushar Krishna*
*Georgia Institute of Technology, geonhwa.jeong@gatech.edu, tushar@ece.gatech.edu
‡Pacific Northwest National Laboratory, {gokcen.kestor, roberto.gioiosa}@pnnl.gov
†IBM Research, prasanth@ibm.com
‖NVIDIA, {aparashar, poant}@nvidia.com
¶Sandia National Laboratories, srajama@sandia.gov

***Abstract*—To meet the extreme compute demands for deep learning across commercial and scientific applications, dataflow accelerators are becoming increasingly popular. While these "domain-specific" accelerators are not fully programmable like CPUs and GPUs, they retain varying levels of flexibility with respect to data orchestration, i.e., dataflow and tiling optimizations to enhance efficiency. There are several challenges when designing new algorithms and mapping approaches to execute the algorithms for a target problem on new hardware. Previous works have addressed these challenges individually. To address this challenge as a whole, in this work, we present a HW-SW co-design ecosystem for spatial accelerators called Union[1] within the popular MLIR compiler infrastructure. Our framework allows exploring different algorithms and their mappings on several accelerator cost models. Union also includes a plug-and-play library of accelerator cost models and mappers which can easily be extended. The algorithms and accelerator cost models are connected via a novel mapping abstraction that captures the map space of spatial accelerators which can be systematically pruned based on constraints from the hardware, workload, and mapper. We demonstrate the value of Union for the community with several case studies which examine offloading different tensor operations (CONV/GEMM/Tensor Contraction) on diverse accelerator architectures using different mapping schemes.**

***Index Terms*—Spatial accelerators, MLIR, Deep learning**

I. Introduction

Numerous custom ASIC accelerators have emerged in the recent past to effectively exploit massive parallelism and locality in the Machine Learning (ML) applications. The most popular examples, such as TPU [1], xDNN [2], RAPID [3], are based on the systolic arrays. There are also more advanced forms including NVDLA [4], Eyeriss [5], ShiDianNao [6] and MAERI [7]. These accelerators have demonstrated lower runtime and higher energy efficiency relative to existing popular architectures such as multi-core CPUs and many-core GPUs [1]. The main architectural features that distinguish these "spatial" accelerators from CPUs and GPUs are parallelism using hundreds to thousands of processing elements (PEs), efficient communication using a fast network-on-chip (NoC) to connect those PEs, and aggressive data reuse using private/shared scratchpad buffers with efficient scheduling.

The success of these accelerators within the context of ML draws researchers' attention to using these accelerators in other compute-intensive domains as well, such as High Performance Computing (HPC) applications. On the other hand, the accelerators are evolving rapidly with novel designs to support new application targets or to provide better performance. Comparing all those novel designs and understanding whether they can be good solutions to a specific algorithm/workload have become incredibly difficult for computer architects and compiler researchers. *Hence, there is a strong need for a flexible, composable, and reusable framework for evaluating new algorithms, their mappings on new spatial accelerator architectures.*

There are three critical components, algorithm/workload, mapping, and hardware for such an ecosystem. In the previous works, these are tightly coupled to each other. For example, simulators [8], [9] and analytical cost models [10], [11] are focusing on a limited set of accelerators. They are also tightly coupled to a set of tensor operations as their inputs. New high level interfaces or new algorithms sometimes require intrusive changes to the cost models. In this work, we develop *unified abstractions* in order to design a modular framework and mitigate the aforementioned problems.

The workload inputs for cost models vary depending on the cost models as well. State of the art cost models require users to translate the operation in a specific format for the cost model [11] or translate a coarse-grained operation into fine-grained operations that the cost model understands [10]. Since this process is different depending on the frameworks, it requires manual efforts by users, which can be error-prone and tedious. A *unified workload abstraction* for the cost models that we are presenting would get rid of this inefficiency. The current cost models also differ in the mapping abstractions. For example, MAESTRO [10] uses data-centric mapping, Interstellar [12] uses Halide scheduling, and Timeloop [11] uses loop-nest mapping. These abstractions have different strengths and limitations in expressing all possible mappings of various

§Joint second authors
[1]https://github.com/union-codesign/union

tensor computations and estimating cost metrics for these mappings on a new accelerator. The existing mappers [11]–[17], which find optimal mappings for the target workload and accelerator, are tightly dependent on their cost models due to the different mapping representations. This limits the interoperability and reusability of the mappers even though conceptually mappers could be used among different cost models if they use a *unified mapping abstraction*. Finally, a *unified hardware abstraction* is needed to represent a broad set of accelerators with diverse interconnects and memory hierarchies [7], [18]–[20] to explore future hardware designs.

This work introduces Union, a unified ecosystem to evaluate tensor operations on spatial accelerators while addressing the challenges mentioned above. The ecosystem is designed with unified abstractions at every level, starting from a tensor operation and its mapping description to hardware description. These abstractions enable the usage of different mappers and cost models interchangeably. Also, these abstractions are generic enough to use for future cost models and mappers. Our ecosystem leverages the recently introduced MLIR infrastructure [21] to integrate with different high-level languages or frameworks, such as Tensorflow, PyTorch for ML, and COMET [22] for HPC. To the best of our knowledge, Union is the first framework unifying multiple high-level frameworks for tensor computations, mappers, and cost models for spatial accelerators. We believe that our work would reduce the burden of computer architects, compiler researchers, and algorithm designers with our unified abstractions and ecosystem. In summary, the contributions of this paper are listed below:

- We provide a plug-and-play unified ecosystem to quickly evaluate tensor operations in various domains such as ML and HPC on spatial accelerators leveraging the MLIR infrastructure.
- We introduce new unified abstractions to describe tensor operations and their mappings on spatial accelerators to integrate different mappers and cost models. This allows us to evaluate diverse tensor operations from HPC kernels and ML use cases.
- We introduce operation-level/loop-level analysis to identify operations to be evaluated with the target spatial accelerator using a cost model.
- We show how our framework can be used with various workloads using different mappers and cost models for the diverse set of accelerators, including flexible and chiplet-based ones. The studies provide an inspiration for the future co-design of tensor operations and spatial accelerators.

We believe Union framework could enhance the co-design opportunities between compiler researchers, algorithm developers, computer architects and simulation tool developers.

II. Background

A. Tensor Operations

In this section, we discuss several key tensor operations across ML and HPC.

Algorithm 1: A loop nest for a CONV2D Operation

Input: *IA*: An input activation with $[n][c][x][y]$
OA: An output activation with $[k][c][x'][y']$
F: An array of filters with $[k][c][r][s]$
stride: Stride for sliding windows

```
for n = 0 to N-1 do
  for k = 0 to K-1 do
    for x = 0 to (X-R) / stride do
      for y = 0 to (Y-S) / stride do
        for c = 0 to C-1 do
          for r = 0 to R-1 do
            for s = 0 to S-1 do
              xx = x × stride + r
              yy = y × stride + s
              OA[n][k][x][y]+ =
                IA[n][k][xx][yy] × F[k][c][r][s]
```

Algorithm 2: A loop nest for a TC Operation

Input: A: A 4D input tensor with $[d][f][g][b]$
B: A 4D input tensor with $[g][e][a][c]$
C: A 6D output tensor with $[a][b][c][d][e][f]$

```
for a = 0 to A-1 do
  for b = 0 to B-1 do
    for c = 0 to C-1 do
      for d = 0 to D-1 do
        for e = 0 to E-1 do
          for f = 0 to F-1 do
            for g = 0 to G-1 do
              C[a][b][c][d][e][f]+ =
                A[d][f][g][b] × B[g][e][a][c]
```

Deep Neural Network (DNN) Models. Recently, DNN models are outperforming other ML conventional techniques in various domains. Convolution layers and fully-connected layers form the bulk of most DNN models, with the former dominating computer vision models and the latter dominating Natural Language Processing (NLP) and recommendation models. From an acceleration perspective, the 2D convolution (CONV2D) and generalized matrix-multiplication (GEMM) operations are being widely used to represent these two layers respectively. The algorithm 1 describes the convolution operation using the loop nest representation. Some accelerators such as TPU [1] use algorithmic transformations such as the *im2col* [26] to convert CONV2D to the GEMM operation while others directly compute convolution operations.

HPC Kernels. Tensor Contraction (TC) operations are generalization of matrix multiplications with arbitrary dimensions. They are popular in HPC domains including many scientific and engineering problems, such as quantum chemistry and finite-element methods. For example, the perturbative triples correction in couple cluster CCSD(T) [27] methods used in the NWChem computational chemistry framework [28], [29] produces a 6D output tensor from two 4D inputs tensor. The

TABLE I
COMPARISON OF OUR FRAMEWORK UNION WITH OTHER EXISTING FRAMEWORKS.

Framework	Target hardware	Cost models	Mappers	Operation abstraction	Mapping abstraction	Hardware abstraction	Integration with high-level frameworks	Target usecase
AutoSA [23]	Systolic	Custom	Custom	Polyhedral models	Space-Time projections	Custom format	N/A	Co-design
MAESTRO [10]	Spatial	Generic	Marvel, GAMMA	Fixed operations	Cluster-target Data-centric	3-level accelerators	Custom parser from TF, PyTorch (ML)	Co-design
Timeloop [11]	Spatial	Generic	Mind Mapping, Random-based, Brute-force	Nested loops	Memory-target Loop-centric	Hierarchical	Custom parsers from TF (ML)	Co-design
Interstellar [12]	Spatial	Generic	Heuristics	Fixed operations	Halide scheduling	3-level accelerators	N/A	Co-design
XLA	TPU	Custom	Custom	LHLO	LHLO	Specific to TPU	TF (ML)	Compilation
ZigZag [24]	Spatial	Generic	Heuristics, LOMA	Nested loops	Memory-target Loop-centric	Hierarchical	N/A	Co-design
XLA	TPU	Custom	Custom	LHLO	LHLO	Specific to TPU	TF (ML)	Compilation
TVM [25]	Specific (e.g., VTA)	Generic	Annealing	TVM statements	TVM scheduling	Specific to target	TF, ONNX (ML)	Compilation
Union	Spatial	Generic	Unified	Nested loops	Cluster-target Loop-centric	Hierarchical	TF (ML), COMET (HPC)	Co-design

corresponding loop nest is shown in algorithm 2. Tensor contractions are computationally intensive and dominate the execution time of many computational applications, thus many optimizations have been developed to improve the performance of executing these kernels. Traditional compilers mostly focus on optimizations such as loop fusions, loop tiling, and loop reordering. High-level Domain-Specific Language (DSL) compiler, instead, can take advantages from re-formulating tensor contractions in a form that is amenable for execution of heterogeneous devices. For example, the COMET compiler [22], a DSL compiler for dense and sparse tensor algebra for chemistry and graph analytics, reformulates tensor contractions by rewriting them with equivalent transpose-transpose-GEMM-transpose (TTGT) expressions. The TTGT computation first flattens the tensors into matrices via explicit tensor transposition and reshape operations, then executes GEMM, and finally folds back the resulting matrix into the original tensor layout. The main advantage of this reformulation comes from leveraging highly efficient GEMM accelerators such as the NVIDIA tensor core [30] or other novel dataflow accelerators, such as the ones targeted in this work. These advantages usually overcome the additional transpositions and generally yield higher performance. However, rebuilding the semantics of a tensor contraction from optimized loops is complicated. To achieve high performance on novel dataflow architectures, it is paramount that a compiler retains the semantics of the language operations throughout all the optimization steps, which explain why most of the novel dataflow accelerators proposed leverage high-level languages.

B. Multi-Level Intermediate Representation (MLIR)

To bridge the semantic gap between high-level language and low level Intermediate Representations (IRs), we leverage the MLIR framework. MLIR has been proposed for both reusability and extensibility [21] and allows intergration of multiple IRs with different level of semantics at the same time.

Currently, many languages and libraries exist, including TensorFlow, Rust, Swift, and Julia, that rely on their own specific IR. On the other hand, multiple target architectures are emerging, especially in the Artificial Intelligence (AI) domain. Maintaining all these compiler frameworks and porting each of them to any new architecture are challenging tasks, which may limit the scope of each language to a limited number of target architectures. The MLIR framework addresses this fragmentation problem by proposing a modular and reusable IR stack that sits in between the language representation and the architectural representation [21]. In this way, architectural specific operations and types can be encapsulated in specific IRs, while sharing common operations, types, and optimizations across languages and target architectures.

MLIR also supports the compilation of high-level abstractions and domain-specific constructs while providing a disciplined and extensible compiler pipeline with gradual and partial lowering. The design of MLIR is based on minimal fundamental concepts and most of the IRs in MLIR could be fully customized. Users can build domain-specific compilers and customized IRs, as well as combining with existing IRs, opting in to optimizations and analysis. The core MLIR concepts include the followings.

- **Operations** are the units of semantics and model concepts from "instructions" to "functions" and "modules". An operation always has an unique opcode. It takes arbitrary number of static single assignment (SSA) operands and produces results. It may also have attributes, regions, blocks arguments, and location information as well.
- **Attributes** provide compile-time static information, such as integer constant values, string data, or a list of constant floating point values.

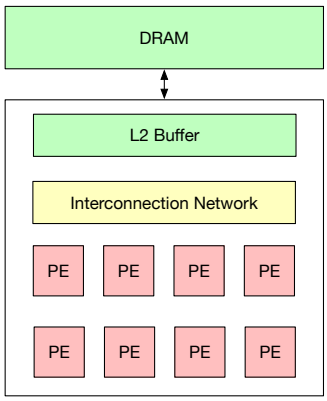

Fig. 1. A simple spatial accelerator architecture with 8 PEs.

- **Values** are the results of an operation or block arguments, and a value always has a type defined by the type system. A *type* contains compile-time semantics for the value.
- **Dialects** consist of a set of operations, attributes and types which are logically grouped and work together.
- **Regions** are attached to an instance of an operation to provide the semantics (e.g., the method of reduction in reduction operation).

Moreover, a region comprises a list of blocks, and a block comprises a list of operations. Beyond the built-in dialect in MLIR system, MLIR users can easily define new dialects, types, operations, analysis or transformation passes and so on. This feature makes MLIR easily extensible.

In this work, we leverage MLIR to decouple high-level language semantics, general optimizations and transformations, and architecture-specific mappings focusing on operations, attributes, and dialects.

C. Spatial Accelerators

To increase the compute throughput while achieving high energy-efficiency for DNN operations, various spatial accelerators have been proposed recently from both industry and academia. A simple spatial accelerator architecture composed of eight PEs with shared L2 buffer are shown in Fig. 1.

*1) **Architecture**:* The spatial accelerators can be categorized into three groups based on their structure: rigid accelerators (e.g., TPU [1], NVDLA [4], Eyeriss), flexible accelerators (e.g., Eyeriss_v2 [19], MAERI [7], SIGMA [31]) and multi-chiplet accelerator (e.g., Simba [18]). Unlike traditional architectures including CPUs and GPUs, spatial architectures use scratchpads as their intermediate buffers. Scratchpads are programmable so that the user can stage intermediate data tiles to maximize data reuse by properly *mapping* the data at the right time at the right location.

*2) **Cost Models**:* To quickly evaluate the performance and energy-efficiency of accelerators, the architecture community has been developing various cost models. Unlike CPUs and GPUs, where runtime contention for shared resources in the datapath and memory hierarchy can lead to non-determinism, accelerators can actually be modeled to the fairly accurate degree as their datapaths and memory hierarchies are tailored to the operation they are designed to accelerate. This allows accelerators to be modeled analytically without requiring cycle-level simulations. Different cost models exist today in the community for modeling different kinds of accelerators at varying degrees of fidelity. For e.g., SCALE-sim [8] models systolic arrays (e.g., Google TPU), MAESTRO [10] models spatial arrays with configurable aspect ratios [7], [19], Timeloop [11] can model hierarchical spatial arrays with complex memory hierarchies (e.g., partitioned buffers and buffer bypassing [20]), and Tetris [32] can model 3D arrays.

*3) **Mappers**:* Using an accelerator cost model, one can estimate the performance of the program with a specific mapping on the target hardware. However, it is not straightforward to find the optimal mapping for a given workload and an architecture for two reasons. First, the space of mappings can be extremely large [11] which makes exhaustive searches infeasible. This has led to several mappers being developed to reduce the search time by pruning the search space or searching with efficient methods. Marvel [13] proposes a decoupled approach to decouple the off-chip map-space from the on-chip one, Timeloop [11] leverages sampling-based search methods, Interstellar [12] uses heuristic-based search, Mind Mapping [33] develops a surrogate model to perform gradient-based search, and GAMMA [15] uses genetic-algorithm based method to efficiently progress by leveraging the previous results. This is currently an active area of research and we expect many more to come. Next, defining the map-space can often be complex by itself since different operations and diverse hardware accelerators may impose constraints on the mappings that are feasible. This is the reason why the mappers described above are highly tied to specific cost models today, limiting interoperability. We discuss this further in the following section.

D. Challenges with Existing Frameworks

The main challenge of the existing frameworks is that they have been developed in a tightly-coupled manner. For example, MAESTRO is a cost model which estimates the performance of the hardware only when a mapping is given. Therefore, it does not find an optimal mapping for the hardware for a workload. GAMMA and Marvel are mappers which search for the optimal mapping for the target hardware/workload using MAESTRO as the cost model. Since both GAMMA and Marvel are tied to MAESTRO, it is not possible to reuse mappers in GAMMA and Marvel using another cost model like Timeloop without having non-trivial engineering effort. On the other hand, Timeloop includes both a cost model and a mapper. Similar to the previous example, it is not possible to use MAESTRO as the backend cost model using the Timeloop's mapper without significant engineering effort. We summarize the comparison of our Union framework with prior frameworks in Table I. Since the goal of our work is to bring such Accelerator Design-Space Exploration tools under

a unified framework, to the best of our knowledge, there is no such framework to compare directly with our approach.

Unfortunately, the lack of interoperability stifles innovation, since none of the mappers and cost models is perfect. Most new accelerator proposals develop new cost models for their design, but they are only able to demonstrate their efficiency for a few hand-optimized mappings. Similarly, researchers working on mapping/compilation for accelerators typically evaluate its efficiency on a specific accelerator for which they have access to the specific cost model (or real hardware).

This problem gets exacerbated as we move up the software stack since DNN model developers using high-level frameworks rely on very simple metrics like total number of Multiply-Accumulate (MAC) operations or the number of trainable parameters in their model to estimate the efficiency of the model which has been shown to be ineffective and oftentimes misleading [34] as it loses all nuances related to the dataflow of the accelerator and data reuse capabilities.

We believe it is crucial to enable domain-experts, compiler experts, and computer architects to have an access to an end-to-end infrastructure that provides a library of plug-and-play mappers and cost models so that users can explore different options interchangeably, and focus on their specific research target (e.g., a new DNN model or a new mapper or a cost model for a new accelerator) without having to engineer or approximate the other parts. Considering the features we discussed previously, MLIR can play a role as the right bridge for this effort.

III. Overview of UNION

In this section, we describe our framework, Union. The overview of the framework is shown in Fig. 2. A user will use Union by specifying workload in high-level language like TensorFlow or DSL, target hardware (with an architecture file and a mapping constraint file), and optimizer options including mapper type, cost model type and unit operation. Union analyzes and lowers the given problem to a Union problem which is used for finding an efficient Union mapping that captures how the data should be tiled and delivered within the memory hierarchy. The affine dialect annotated with a Union mapping can further be lowered to accelerator specific configurations to run the specific accelerators, but this is not the scope of this project and we leave it to the users who want to run their own accelerators. One of the key contributions of Union is a set of abstractions for problem/hardware/mapping to unify different modules which will be presented in Sec. IV. Here, we introduce the overview of Union.

A. Frontend: Using MLIR as a Bridge

To demonstrate the composability and flexibility of our framework, we consider two high level DSLs which target very different application domains, TensorFlow for ML and COMET DSL for computational chemistry.

*1) **TensorFlow**:* TensorFlow is one of the most famous open source platforms for machine learning. Although several independent efforts exist to explore different ways of

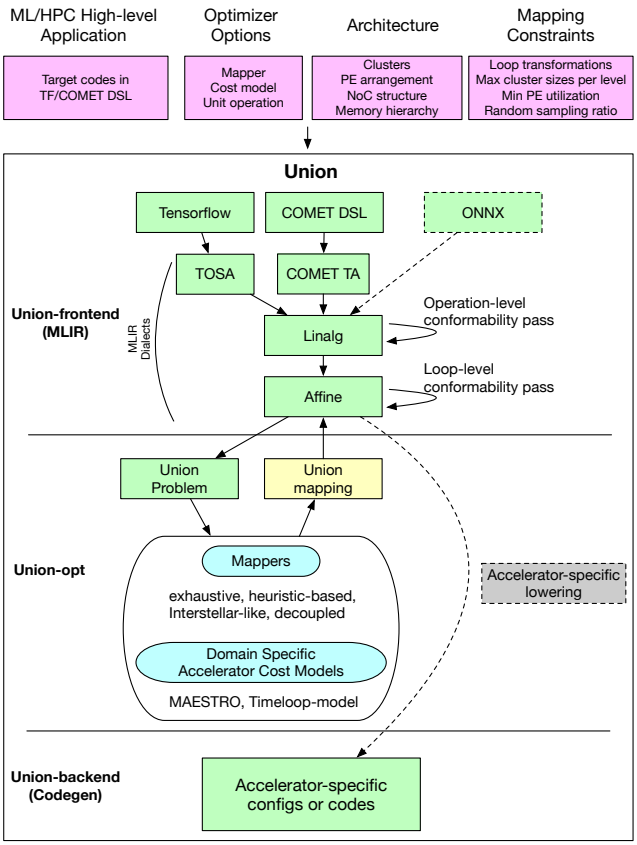

Fig. 2. Union overview. The pink boxes indicate the inputs of Union while green boxes are showing how the codes are getting lowered. Rectangles and arrows with dotted lines are out of the scope of this paper.

lowering TensorFlow code to mid-level MLIR dialects (such as linear algebra) including IREE [35] and NpComp [36], we follow the Tensor Operator Set Architecture (TOSA) dialect approach [37] in this work. Moreover, current efforts, including ours, mostly focus the inference side and assume that the machine learning model has been already built and trained. This approach is common on state-of-the-art DNN accelerators, such as the NVIDIA Deep Learning Accelerator (NVDLA) [4], where models are trained on GPUs and the NVDLA is used for inference.

We use a trained machine learning model using the standard execution flow on CPU, GPU, or TPU, which is saved as a graph. The graph is associated with a set of properties, including shape, types, and number of layers. Next, the generated graph is optimized and converted to its functional counterpart by removing some of the specific TensorFlow information, such as TensorFlow control regions and islands. At this point, this graph can be converted to the TOSA dialect, which is a generic MLIR dialect for tensor algebra targeting machine learning applications. The TOSA dialect is also the lowest domain-specific dialect in our framework. As explained next, the rest of the compilation pipeline including mappers and cost models are shared across the various domains.

*2) **COMET DSL**:* The COMET compiler [22], [38] supports the COMET DSL for sparse and dense tensor algebra computations, focusing on computational chemistry kernels in NWChem and graph analytics. The compiler is based on MLIR [21] which performs a progressive lowering process to map high-level operations to low-level architectural resources. It also includes a series of optimizations performed in the lowering process, and various IR dialects to represent key concepts, operations, and types at each level of the MLIR. At each level of the IR stack, COMET performs different optimizations and code transformations. Domain-specific, hardware-agnostic optimizations that rely on high-level semantic information are applied at high-level IRs. These include reformulation of high-level operations in a form that is amenable for execution on heterogeneous devices (e.g., rewriting TC operations as TTGT) and automatic parallelization of high-level primitives (e.g., tiling for thread- and task-level parallelism). Currently, the compiler generates efficient code for traditional central processing unit (CPU) and GPU architectures as well as Verilog code for FPGAs.

*3) **Lowering to Linalg/Affine Dialect**:* Regardless of the language used for the original application, we lower the code down to the language-specific description of the application to frontend MLIR dialects, e.g., TensorFlow to TOSA or COMET DSL to COMET Tensor Algebra (TA) dialect. This is shown in Fig. 2. Next, we further lower from the domain-specific dialects to generic, language-independent constructs and operations, such as CONV2D and GEMM. In our framework, both TOSA and COMET TA dialects are lowered to a common Linear Algebra (Linalg) MLIR dialect. At this stage, the IR is effectively decoupled from the original language and we can analyze the operations independently from the language. Depending on the accelerator cost model (as discussed next), the operations may be lowered further to Affine dialect for a loop-nest representation.

Cost Model Dependent Conformability Passes. The next step needs to consider the requirements from the underlying cost models. Here, Union transforms and annotates the generic IR obtained in the previous step with information that is necessary for the mapping design space exploration. The cost models we consider in this work targeting spatial accelerators have different constraints for the workloads that they can evaluate. For e.g., MAESTRO natively supports CONV2D and GEMM operations, where as Timeloop supports perfectly affine nested loops with no conditionals. Hence, our framework includes operation-level or loop-level conformability passes to check if the tensor operation is conformable to the underlying cost model for evaluation. These conformability passes embody different constraints (such as checking for specific operations or loop bounds [13]) of different cost models to determine whether it can be evaluated by the cost models.

B. An Optimizer for an Efficient Mapping: Union-opt

After processing at the Union-frontend, the target problem is translated into an instance of Union problem abstraction (Sec. IV). The Union optimizer, Union-opt, searches for the efficient mapping of the problem based on the target metric such as latency, energy, Energy-Delay-Product (EDP). To do so, Union-opt explores the *map-space* for the given problem, architecture and constraint. Different mappings can incur different PE utilization, data distribution, reduction, and data reuse affecting to the performance and energy efficiency. Fig. 3 shows how the latency and energy consumption can vary for different mappings that Union explores for a layer from DLRM [39] on a simple spatial architecture with 16×16 PE array. We will discuss more about how Union-opt can be used through the case studies in Sec. V. Since the mapping space for a simple problem can be extremely large due to the exponential and multiplicative characteristics of number of cases, it is inevitable to have efficient mappers other than exhaustive search [11].

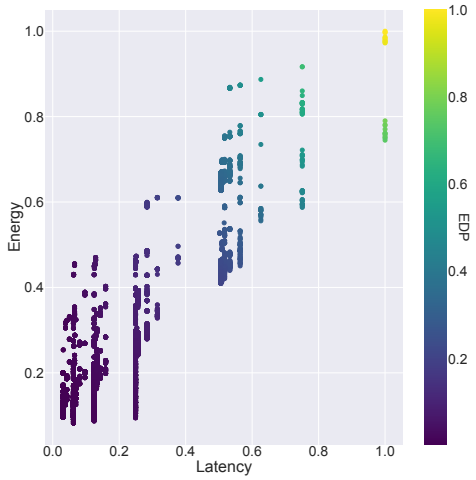

Fig. 3. Normalized energy consumption and latency with EDP for different mappings of a layer from DLRM on a 3-level spatial architecture with 16×16 PE array.

*1) **Mappers**:* We currently integrate a few mappers in Union including exhaustive search, random sampling based search (from Timeloop [11]), decoupled approach (from Marvel [13]) and a few heuristic-based approaches. Users can add their own mappers and/or cost models by supporting our abstractions directly or adding converter from their format to our abstractions (Sec. IV).

*2) **Domain-Specific Accelerator Cost Models**:* We currently implement Timeloop and MAESTRO as the cost models in Union to evaluate the mappings for proof of concept, but Union can easily be integrated with other cost models.

MAESTRO [10] takes a high-level DNN operation such as CONV2D, GEMM, and DWCONV as an input problem. Therefore, whether the given problem is conformable or not depends on the high-level operation type. On the other hand, Timeloop [11] can take a fully nested loop which satisfies a few rules as an input problem. The fully nested loop should have affine indexes and every loop re-ordering should not change the result of the problem. Furthermore, each cost model assumes an unit operation for a PE such as two-operand MAC with certain data type. To evaluate the performance of

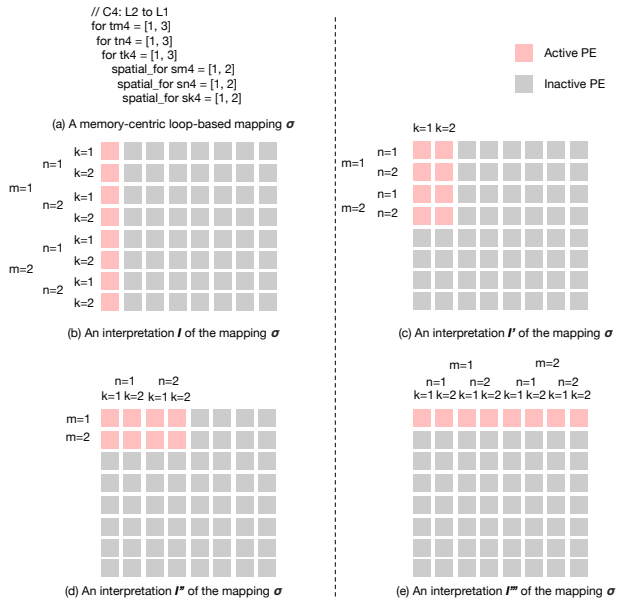

Fig. 4. An example of a memory-target loop-centric mapping and its interpretations on a 8×8 2D PE array.

the given problem, the unit operation should be supported in its energy model. For example, CONV2D can be used as an input problem for Timeloop since it can be described as a fully nested loop following the given rules as shown in algorithm 1 assuming that the energy model is configured with two-operand MAC as its unit operation. Similarly, GEMM or Tensor Contraction can be evaluated using the Timeloop cost model since they all can be described as a fully nested loop following the given rules and using the two-operand MAC operation as its unit operation. Matricized tensor times Khatri-Rao product (MTTKRP) operation cannot be evaluated using Timeloop if its energy model is configured with two-operand MAC as its unit operation, but it can be done by changing three-operand-multiply-add as its unit operation and provide the necessary energy model.

The backend of Union can be customized for generating configurations of individual accelerator targets. The backend is beyond the scope of this paper and part of our future work.

IV. Union Abstractions for Workload, Architecture, and Mapping

Evaluating a mapped problem on a target spatial architecture requires abstractions between the architecture, mapping, and the workload. Different frameworks that evaluates spatial accelerators have come up with different abstractions respectively to compare the performance and energy consumption of mappings of tensor operations on spatial architectures. We first discuss some of their limitations and present the abstractions we developed for Union.

A. Limitations of Current Mapping Abstractions

1) ***Memory-target Loop-centric Approach****:* Most of previous frameworks [11]–[15] use each hardware memory level as the target of a loop tiling level (i.e. tiling can only happen in each memory hierarchy level, such as between L2 and L1 buffers) to exploit temporal and spatial locality. Fig. 4 shows a memory-target loop-centric mapping $\boldsymbol{\sigma}$ and four different possible interpretations of such mapping on a 8×8 2D PE array. Its loop nest representation is shown in Fig. 4(a) where `for` loops describe the temporal mapping while `spatial_for` loops describe spatial mapping (i.e., parallel units). Since the $\boldsymbol{\sigma}$ does not have the information about in which direction the problem dimensions is parallelized in physical spatial units, the mapping can be realized by all options in Fig. 4(b)-(e). To circumvent the ambiguity the mapping representation, prior frameworks either assume certain implicit rules specific to the accelerators, or introduce extra annotations to indicate the mapping with spatial distribution and physical spatial axis. Another limitation of such abstraction is that there is a 1-to-1 mapping between a tensor rank and physical spatial dimension in the memory-target representations. For example, memory-target abstraction cannot describe parallelizing the M dimension onto both horizontal and vertical axis in the PE array. Similarly, it is impossible to precisely describe a mapping which distributes M and N dimensions on horizontal axis and distributes N and K dimensions on the vertical axis using memory-target loop-centric mapping scheme. Moreover, due to the hierarchical order between `spatial_for` loops, two iterators cannot change concurrently except at the loop bounds. Such limitation forbids a mapping which parallelizes different problem dimensions at the same time.

2) ***Cluster-target Data-centric Approach****:* MAESTRO [10] introduces the notion of clusters. A cluster means a *logical* group of PEs. Instead of fixed hardware memory levels, MAESTRO targets each logical cluster level for tiling to explore more fine-grained tiling opportunities and remove the ambiguity caused by memory-target approach. However, MAESTRO's mapping abstractions use a *data-centric* notation which is not suitable to reason about using high-level compute-based abstractions, such as MLIR. Moreover, MAESTRO assumes a *fixed* accelerator architecture: a 2-level memory hierarchy with private L1 buffers and shared L2 buffer, so it is not possible to explore mappings for accelerator architectures with more complex memory hierarchy.

Union combines the best of both these approaches discussed above and introduces a *logical cluster-target loop-centric approach*. We adopt a cluster-target approach to be able to describe more mappings (addressing the shortcoming in Timeloop's representation) while still enabling a straightforward translation between our notation and loops from MLIR (addressing the shortcoming in MAESTRO's representation). Table II shows the differences between prior abstractions and Union.

B. First Abstraction: From MLIR Dialects to a Problem Instance

Common cost models support a set of workloads, defined in different levels (ex. operation level for MAESTRO [10] while loop level for Timeloop [11]). From the workload written in

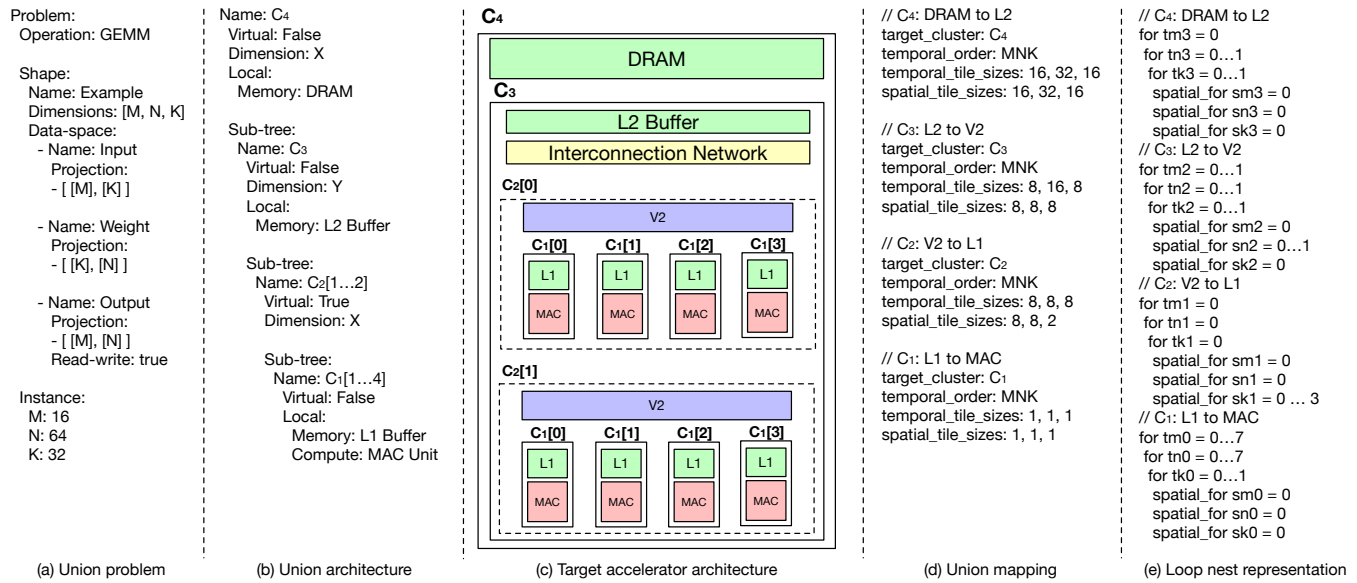

Fig. 5. Union abstractions to describe a GEMM problem with a mapping on an accelerator which is composed of a simple 2D PE array. (a) describes a Union problem for a GEMM problem and (b) describes the Union architecture of the target architecture shown in (c). (d) shows a Union mapping that shows how to map the data to the architecture to run the GEMM problem. (e) represents the Union mapping in loop nest form.

TABLE II
COMPARISON BETWEEN PRIOR ABSTRACTION AND UNION.

	Hardware Memory-target	**Logical Cluster-target**
Data-centric	N/A	MAESTRO [10]
Loop-centric	Timeloop [11], Interstellar [12]	Union (**This work**)

high-level language and the target cost model, Union-frontend extracts the information from both levels as an affine dialect with an operation annotation. To handle the given problem with an affine dialect with an operation annotation, our abstractions includes loops, projections of the data spaces from array references, and operation type as shown in Fig. 5(a)[2]. Loop iterators in the affine loop are set as dimensions and array references set each data in data-space with their projections. Finally, the size of each dimension is derived from the loop bounds. The affine dialect is analyzed and re-organized to set dimensions, data-space, projection, and instance. We use the attribute Operation to indicate the operation (if given). This abstraction captures both operation-level and loop-level information so that any cost model which supports one of them can be used.

C. Second Abstraction: Describing Architecture

One of the key features of Union is to describe a logical spatial architecture instead of a fixed one. We start with the hierarchical architecture abstraction used in the previous work [11] and extend it to describe the architecture in the logical cluster-target manner. Fig. 5(b) shows a Union architecture abstraction for the target spatial architecture illustrated in Fig. 5(c). The target architecture is composed of a 2D PE array, an L2 buffer shared across all PEs, and a private L1 buffer for each PE. We call the top cluster level in the n-level cluster architecture as C_n in this paper. For example, in Fig. 5(c), we call the outermost cluster level which has DRAM as its local memory as C_4 while the innermost cluster which has a L1 buffer as its local memory is called as C_1.

Various features can be specified in each cluster level such as compute, memory, and sub-clusters (and size of each sub-cluster). We also add two new attributes in each cluster level, **Virtual** and **Dimension** in addition to the abstractions used in the previous work [11]. The first attribute, Virtual, indicates whether the cluster has a dedicated physical memory or not. The second attribute, Dimension, defines how the sub-clusters are laid in the physical dimension. In the example architecture shown in Fig. 5(c), the cluster at C_4 has DRAM as its memory and is composed of a single sub-cluster as C_3. A cluster at C_3 has L2 buffer and is composed of two instances of C_2 which are laid in Y-axis. A cluster at C_2 is composed of four instances of C_1 which are laid in the X axis. Note that *Virtual* is *True* only for C_2 since C_2 does not have a dedicated memory. Instead, we draw V_2 in the figure which will always be bypassed since it is virtual (imaginary) buffer, but it provides a way to describe the intermediate tiling. The innermost cluster, C_1, includes L1 buffer and a MAC unit. With Union architecture abstraction, one can describe how a multi-level clusters mapped on to multi-dimensional PE arrays. We assume that the parallelism can only be defined across sub-clusters. For example, one can put another virtual cluster between C_2 and C_1 to exploit more fine-grained parallelism. One can also describe partitioned buffer by introducing sibling clusters in the same cluster level, similar to the way how Timeloop [11] describes. Some architectures are limited to

[2]inspired from Timeloop problem instance description

support certain loop orders depending on its dataflow such as input stationary, output stationary, weight stationary or row stationary [5]. Those architectures can be realized by specifying the limitations in the constraint file which we discuss later.

D. Third abstraction: Describing a Mapping between a Problem Instance and a Spatial Accelerator

A mapping describes how a problem instance will be executed on a logical cluster-based architecture. We propose a cluster-target mapping representation using loop-centric approach. Previous loop-based representations [11]–[13] describe the temporal/spatial behavior of tiles in each memory level while our proposed mapping describes the behaviors in each cluster level. In our mapping, the parallelism across sub-clusters can be described at each cluster level. Unlike the memory-target representations, one can describe tiling at a virtual cluster level even though this level does not have dedicated memory units.

Semantics and characteristics. In our mapping abstraction, each tiling level explicitly targets a cluster, not memory, to cover broader mapping variants and remove ambiguity. An example Union mapping is shown in Fig. 5(d) and its loop nest representation is shown in Fig. 5(e). `target_cluster` defines the cluster level of the following tiling directives. `temporal_order` defines the temporal ordering between dimensions in the cluster level. `temporal_tile_sizes` and `spatial_tile_sizes` defines the size of temporal and spatial tile for each dimension.

The spatial tile sizes defined in $(i+1)$th level cluster, $C_{(i+1)}$, can further be divided into sub-tiles in C_i. The tile can be divided into multiple time steps using temporal tiles in C_i. Each temporal tiles have the size as specified in `temporal_tile_sizes` in C_i A temporal tile in C_i can be divided into smaller spatial tiles and be spatially distributed into multiple instances of sub-clusters $C_{(i-1)}$. Therefore, the parallelism in ith level can be calculated by dividing temporal tile size by spatial tile size. Note that we do not define `spatial_order` in each cluster level. We change the semantic of `spatial_for` so that it can change iterators concurrently in the same cluster level, inspired from MAESTRO data-centric notations [10].

Finally, Union introduces a few rules to check if a mapping is legal for the target logical architecture and the problem instance as shown in the following.

- The mapping will be illegal if the spatial tile size of the problem dimension d at ith cluster level is smaller than the temporal tile size of the same problem dimension d at $(i-1)$th cluster level.
- The parallelism for the problem dimension d at ith cluster level, which can be derived as $\frac{TT_d^i}{ST_d^i}$ should be equal to or smaller than the number of $(i-1)$th clusters in a ith cluster level.
- If a ith cluster is not virtual, the size of its memory should be as large as the memory sizes required by temporal tile sizes. TT_d^i and ST_d^i are the temporal and spatial tile size of problem dimension d in ith cluster respectively.
- The mapping should cover all the iteration vectors defined by the problem.

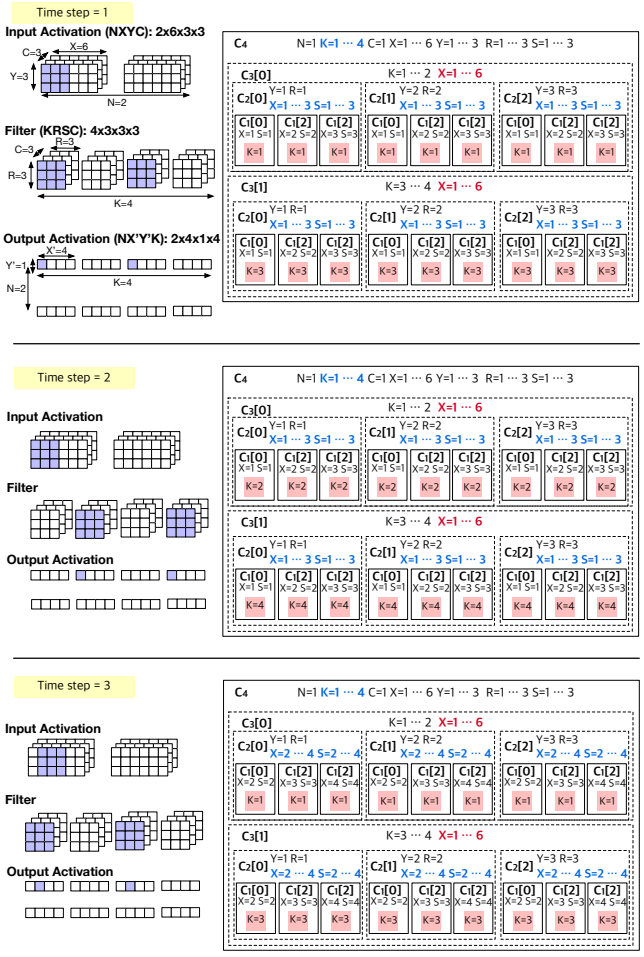

Fig. 6. The visualization of a *K_YR_XS* partitioned mapping for CONV2D using with 18 PEs for 3 time steps. A cluster containing DRAM, C_5, is not shown here. A red box indicates a MAC unit.

Walk-through Example. Fig. 5(d) shows a Union mapping and Fig. 5(e) describes the mapping using the loop nest representation. A Union mapping can describe multi-level clusters of multi-dimensional PE arrays precisely and specify temporal and spatial tiling of data at each level. Also, one can distribute different problem dimensions at the same time in each cluster level over sub-clusters, i.e. there is no temporal ordering between spatial_fors in the same cluster level. A complex example mapping for a small CONV2D operation using a flexible accelerator (such as MAERI [7]) is illustrated in Fig. 6 and the corresponding Union mapping and loop nest representation are shown in Fig. 7. In Fig. 6, the right column shows which data dimension values are mapped onto each cluster. Clusters with solid lines (C_4 and C_1) have the dedicated memory in the level while clusters with dotted lines (C_3 and C_2) do not. We use L2 and L1 to indicate

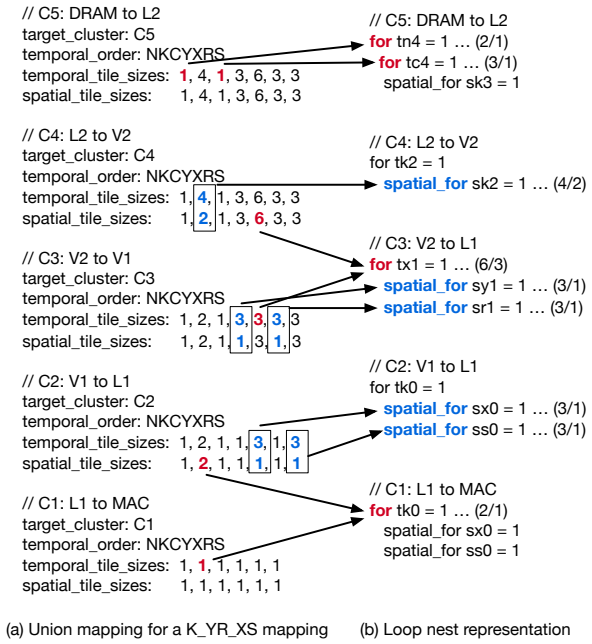

Fig. 7. A Union mapping of a *K_YR_XS* partitioned mapping and its loop nest representation. The order of dimensions in tile_sizes is NKCYXRS.

the memory in C_4 and C_1 and V2 and V1 to indicate the virtual (imaginary) memory in C_3 and C_2. The left column of Fig. 6 visualizes the mapped elements in input activation, filter, and output activation. Purple elements are mapped onto MAC units at each time step. In this mapping, dimension K is spatially distributed across C_3 clusters, Y and R are spatially distributed together across C_2, and X and S are spatially distributed together across C_1. We call this mapping as a *K_YR_XS* partitioned mapping to show the parallelism. Each C_2 cluster is assigned for a row of a channel of a filter and a row of a channel of a input activation and each column in the row will be processed in the C_1 clusters concurrently. Each C_3 cluster is assigned for a channel of a filter and the corresponding input activations. As a result, inputs and outputs are reused between time step 1 and 2 while fetching different filters from the upper memory levels. Between time step 2 and 3, a part of input activations are reused in MAC units and others are being fetched from the upper memory levels.

E. Constraint File

In addition to Union abstractions, a user can also provide *constraints* derived from a specific accelerator, such as feasible tile sizes, loop orders, parallelizing dimensions, and aspect ratio. Such constraints provide the framework extra rules to eliminate illegal mappings and/or prune the mapping space for specific accelerators. For example, to describe a fully flexible accelerator like MAERI, the user will not provide constraint file to describe the hardware. On the other hand, a NVDLA-style [4] architecture can be realized by having a constraint file that forces parallelization on dimensions C and K for a convolution operation with a fixed aspect ratio. Furthermore, a user can set some constraints to prune the map space based

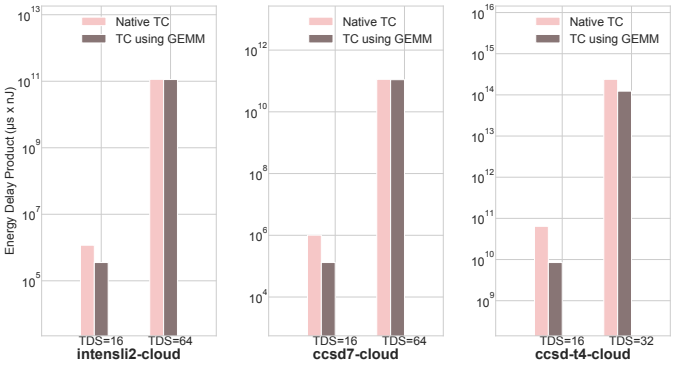

Fig. 8. Three tensor contraction examples with different dimensions using different algorithms (native and TTGT) on a cloud accelerator. We explore Tensor Dimension Sizes (TDS) with 16 and 64 for intensli2 and ccsd7, and 16 and 32 for ccsd-t4.

on min/max PE utilization or specific loop orders or tile sizes that the user wants to explore.

V. Case Studies using Union

In this section, we show three case studies for algorithm exploration, mapping exploration, and hardware exploration to illustrate how Union can be used by domain experts, compiler researchers, and hardware architects, respectively. We evaluate two types of accelerators, edge and cloud, as shown in Table V. We assume 1GHz as the clock frequency and 8 bits for the wordsize with uint8 MAC units. In Union, we directly use Timeloop and MAESTRO, which are already validated against RTL for different existing accelerators. Thus, the validation of the performance numbers is dependent on the fidelity of the underlying cost models.

We choose tensor contractions from the TCCG benchmark suite [40], using the reference problem sizes. The input sets are taken from different domains, including machine learning, molecular dynamics, and quantum chemistry. We use 16, 32, 64 as the Tensor Dimension Sizes (TDS) and assume that every dimension has the size as TDS in a TC problem instance. We use a few representative DNN layers from the MLPerf benchmark including ResNet50 for computer vision, DLRM for recommendation, and BERT for natural language processing. We use N as a batch size, and K, C, X, Y, S, R, NIN, NON as the number of filters, input channels, input cols, intput rows, filter columns, filter rows, input neurons, output neurons. We summarize the TC and DNN workloads that we use in the case studies in Table III and Table IV, respectively.

A. Algorithm Exploration

A single tensor operation can be computed via several algorithms. The Union-frontend determines whether to run an operation natively, or transform it to other operations, depending on which algorithm provides better performance on the accelerator. We demonstrate this use case using tensor contraction running on a cloud type 2D spatial accelerator via two algorithms: (1) running natively and (2) running through TTGT. We use the Timeloop cost model and a mapper based

TABLE III
TENSOR CONTRACTION PROBLEMS AND THE CORRESPONDING GEMM DIMENSION SIZES FOR TTGT

Name	Equation	Tensor Dimension Sizes	GEMM Dimension Sizes
intensli2	C[a, b, c, d] = A[d, b, e, a] × B[e, c]	a = b = c = d = e = 64 a = b = c = d = e = 16	M = 262144, N = 64, K = 64 M = 4096, N = 16, K = 16
ccsd7	C[a, b, c] = A[a, d, e, c] × B[e, b, d]	a = b = c = d = e = 64 a = b = c = d = e = 16	M = 4096, N = 64, K = 4096 M = 256, N = 16, K = 256
ccsd-t4	C[a, b, c, d, e, f] = A[d, f, g, b] × B[g, e, a, c]	a = b = c = d = e = f = g = 32 a = b = c = d = e = f = g = 16	M = 32768, N = 32768, K = 32 M = 4096, N = 4096, K = 16

TABLE IV
DNN LAYER DIMENSIONS USED IN EVALUATION

Layer	Dimensions
ResNet50-1	N=32 K=C=64 X=Y=56 R=S=1
ResNet50-2	N=32 K=C=64 X=Y=56 R=S=3
ResNet50-3	N=32 K=512 C=1024 X=Y=14 R=S=1
DLRM-1	N=512 NIN=1024 NON=1024
DLRM-2	N=512 NIN=1024 NON=64
DLRM-3	N=512 NIN=2048 NON=2048
BERT-1	N=256 NIN=768 NON=768
BERT-2	N=256 NIN=3072 NON=768
BERT-3	N=256 NIN=768 NON=3072

TABLE V
ACCELERATOR CONFIGURATIONS

Type	# of PEs	L1 Buffer Size	L2 Buffer Size	NoC Bandwidth
Edge	256	0.5 KB	100 KB	32 GB/s
Cloud	2048	0.5 KB	800 KB	256 GB/s

```
// C4: DRAM to L2
target_cluster: C4
temporal_order: ECABD
temporal_tile_sizes:  16, 16, 16, 16, 16
spatial_tile_sizes:   16, 16, 16, 16, 16

// C3: L2 to V2
target_cluster: C3
temporal_order: ECABD
temporal_tile_sizes:  16, 1, 16, 1, 16
spatial_tile_sizes:   1, 1, 16, 1, 16

// C2: V1 to L1
target_cluster: C2
temporal_order: ECABD
temporal_tile_sizes:  1, 1, 16, 1, 16
spatial_tile_sizes:   1, 1, 16, 1, 1

// C1: L1 to MAC
target_cluster: C1
temporal_order: ECABD
temporal_tile_sizes:  1, 1, 1, 1, 1
spatial_tile_sizes:   1, 1, 1, 1, 1, 1
```

(a) Optimal Union mapping found for intensli2 running natively with TDS = 16

```
// C4: DRAM to L2
target_cluster: C4
temporal_order: MKN
temporal_tile_sizes: 4096, 16, 16
spatial_tile_sizes: 4096, 16, 16

// C3: L2 to V2
target_cluster: C3
temporal_order: KMN
temporal_tile_sizes: 4096, 1, 16
spatial_tile_sizes: 4096, 1, 1

// C2: V1 to L1
target_cluster: C2
temporal_order: KMN
temporal_tile_sizes: 4096, 1, 1
spatial_tile_sizes: 64, 1, 1

// C1: L1 to MAC
target_cluster: C1
temporal_order: KMN
temporal_tile_sizes: 1, 1, 1, 1, 1, 1
spatial_tile_sizes: 1, 1, 1, 1, 1, 1
```

(b) Optimal Union mapping found for intensli2 running through GEMM with TDS = 16

Fig. 9. Generated mappings from Union for intensli2 using different algorithms with tensor dimension sizes as 16. The orders of dimensions in tile_sizes are ABCDE and MNK for the mappings in (a) and (b) respectively. Blue tilesizes show the spatial distribution while red tilesizes show the temporal distribution for the dimension.

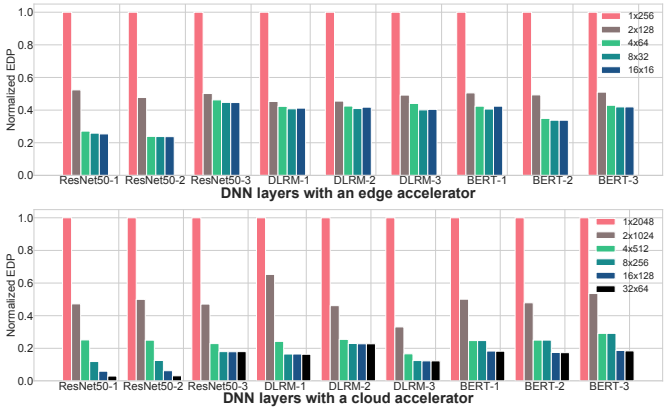

Fig. 10. EDP comparison on DNN workloads using a flexible accelerator with different aspect ratio.

on both heuristic and random sampling. We use the cloud configuration in Table V with 32×64 as the aspect ratio of the accelerator to balance the parallelism across rows and columns. Note that for TTGT cost estimation, the cost model only estimates the cost of the GEMM operation assuming that the cost of transpose operations would not be significant. Since TTGT does not incur duplicated elements of the original tensors, the memory footprint for both running TC natively and running TC with TTGT have the same memory footprint. Fig. 8 plots the Energy-Delay-Product (EDP) for three tensor contractions with tensor dimensions 16 and 64 on the cloud accelerator. We observe that the lower EDP is achieved when running with TTGT for all cases with TDS=16. This is because running natively will under-utilize the available compute units since the target accelerator has 32×64 PEs while the each tensor dimension has size of 16. For example, Fig. 9 shows the mappings generated from Union for Intensli2. In Fig. 9(a) C_3 level, we observe that the optimal mapping found by Union distributes the problem dimension A across 16 C_2s and distributes the dimension E across 16 C_1s, resulting in utilizing 256 PEs with *A_E* partitioned mapping. In Fig. 9(b), the optimal mapping with GEMM distributes K across 16 C_2s and distributes M across 64 C_1s, resulting in utilizing 1024 PEs with *K_M* partitioned mapping.

B. Mapping Exploration

Flexible accelerators like Eyeriss_v2 [19] and MAERI [7] can logically configure to different aspect ratios for the underlying PE array via configurable NoCs. This flexibility allows these accelerators to efficiently run layers with different shapes and sizes. We demonstrate the value of Union by exploring optimized array configurations for different DNN workloads for such flexible substrates. For this case study, we use the

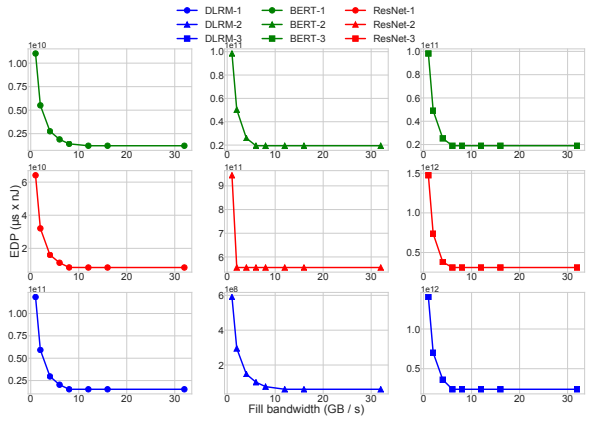

Fig. 11. EDP comparison with different fill bandwidth on multi-chiplet based architecture using DNN workloads.

MAESTRO cost model as it has support to model such flexible accelerators. The flexibility in aspect ratios gets captured by allowing cluster sizes to be variable. In the Union constraint file, we specify different cluster sizes to explore different aspect ratio.

We evaluate the DNN workloads shown in Table IV using different aspect ratio for the edge (1×256, 2×128, 4×64, 8×32, and 16×16) and cloud accelerators (1×2048, 2×1024, 4×512, 8×256, 16×128, and 32×64). Each aspect ratio corresponds to a configuration of the flexible accelerator. Fig. 10 plots the EDP. We observe that the EDP gets saturated once it maximizes the PE utilization after the mapper finds the optimal tile sizes and loop orders to maximize the data reuse. Even though the balanced aspect ratio showed the best performance for most of the cases that we evaluate, this can be sub-optimal if the workload is unbalanced. For example, GEMM with 4×2048 or 2048×2 or 4×2 will be able to fully utilize an accelerator with 1×2048 aspect ratio by parallelizing K dimension while 32×64 accelerator will be underutilized. This is where Union's cluster-centric approach to describe mappings helps as it enables mapping the same workload dimensions to different spatial dimensions to fully exploit the available parallelism.

C. Hardware Exploration

In our last case study, we study the impact of chipletization on an accelerator's performance. Multi-chiplet based architectures are gaining popularity as they can reduce manufacturing cost and provide scalabililty. NVIDIA's Simba [18] is a recent example. However, the inter-chiplet network is more expensive than on-chip network resulting in lower bandwidth and higher energy. For this case study, we use an accelerator which is composed of 16 chiplets, and each chiplet has the same configuration with the edge accelerator in Table V. The total number of PEs are equal to 4096. We study the impact of the interconnect bandwidth by varying the fill bandwidth of the global buffer in each chiplet, i.e. the bandwidth from DRAM to the global buffer in each chiplet. We use Timeloop for this case study as it can model hierarchical architectures like Simba and also comes with the Accelergy [41] energy model for accurately estimating on-chip versus on-package energy.

Fig. 11 plots our results. For all models, we observe that EDP drops rapidly with the increase in fill bandwidth when the fill bandwidth is low, and it gets saturated once the fill bandwidth is sufficient so that it is not bounded by the fill bandwidth. According to the result, different layers get saturated in the different fill bandwidth depending on the available data reuse. We also observe that ResNet-2 gets saturated when fill bandwidth is 2GB/s while others get saturated between 6 - 12 GB/s.

VI. Conclusion

In this work, we propose Union, a unified framework for evaluating tensor operations on spatial accelerators. Our MLIR based framework allows to map both HPC and ML tensor operations using multiple mappers to multiple cost models for spatial accelerators. The three case studies presented demonstrate the flexibility of the framework by evaluating very different operations, mappings, and hardware features with a single framework. While the number of operations, mappings and accelerators are currently limited to what we have demonstrated here, we plan to extend to other kernels such as tensor decomposition, other accelerators and mappers in the near future. There are also advanced features that can be added to Union abstractions to support fused operations, sparsity-aware accelerator cost models, and unimodular/polyhedral mappings, but we leave those as a future work. The modular framework allows us to add such changes without requiring a redesign of the whole software stack.

VII. Acknowledgment

We thank Ruiqin Tian at PNNL for her help with the COMET compiler. We also thank Hyoukjun Kwon, Clayton Hughes, Mark Plagge, Juan Escobedo, Rizwan Ashraf for insightful comments and discussions on this work. We also thank the anonymous reviewers for their valuable feedback. Support for this work was provided through U.S. Department of Energy's (DOE) Office of Advanced Scientific Computing Research as part of the Center for Artificial Intelligence-focused Architectures and Algorithms. PNNL is operated by Battelle for the DOE under Contract DE-AC05-76RL01830. Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA-0003525.

REFERENCES

[1] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P.-l. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, and D. H. Yoon, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*. New York, NY, USA: Association for Computing Machinery, 2017, p. 1–12.

[2] "Accelerating dnns with xilinx alveo accelerator cards," https://www.xilinx.com/support/documentation/white_papers/wp504-accel-dnns.pdf.

[3] B. M. Fleischer *et al.*, "A Scalable Multi- TeraOPS Deep Learning Processor Core for AI Trainina and Inference," in *2018 IEEE Symposium on VLSI Circuits*, 2018, pp. 35–36.

[4] NVIDIA, "The nvidia deep learning accelerator (nvdla)," http://nvdla.org/hw/v1/ias/programming_guide.html.

[5] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," in *Proceedings of the 43rd International Symposium on Computer Architecture*, 2016, p. 367–379.

[6] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam, "Shidiannao: Shifting vision processing closer to the sensor," in *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*, 2015, pp. 92–104.

[7] H. Kwon, A. Samajdar, and T. Krishna, "Maeri: Enabling flexible dataflow mapping over dnn accelerators via reconfigurable interconnects," in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. New York, NY, USA: Association for Computing Machinery, 2018.

[8] A. Samajdar, J. M. Joseph, Y. Zhu, P. Whatmough, M. Mattina, and T. Krishna, "A systematic methodology for characterizing scalability of dnn accelerators using scale-sim," in *2020 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2020, pp. 58–68.

[9] F. Muñoz-Martínez, J. L. Abellán, M. E. Acacio, and T. Krishna, "STONNE: A Detailed Architectural Simulator for Flexible Neural Network Accelerators," *arXiv e-prints*, p. arXiv:2006.07137, Jun. 2020.

[10] H. Kwon, P. Chatarasi, M. Pellauer, A. Parashar, V. Sarkar, and T. Krishna, "Understanding reuse, performance, and hardware cost of dnn dataflow: A data-centric approach," in *MICRO*, 2019.

[11] A. Parashar, P. Raina, Y. S. Shao, Y. Chen, V. A. Ying, A. Mukkara, R. Venkatesan, B. Khailany, S. W. Keckler, and J. Emer, "Timeloop: A systematic approach to dnn accelerator evaluation," in *2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2019, pp. 304–315.

[12] X. Yang, M. Gao, Q. Liu, J. Setter, J. Pu, A. Nayak, S. Bell, K. Cao, H. Ha, P. Raina, C. Kozyrakis, and M. Horowitz, "Interstellar: Using halide's scheduling language to analyze dnn accelerators," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2020.

[13] P. Chatarasi, H. Kwon, N. Raina, S. Malik, V. Haridas, A. Parashar, M. Pellauer, T. Krishna, and V. Sarkar, "Marvel: A data-centric compiler for dnn operators on spatial accelerators," *arXiv preprint arXiv:2002.07752*, 2020.

[14] S. Dave, Y. Kim, S. Avancha, K. Lee, and A. Shrivastava, "Dmazerunner: Executing perfectly nested loops on dataflow accelerators," *ACM Trans. Embed. Comput. Syst.*, Oct. 2019.

[15] S.-C. Kao and T. Krishna, "Gamma: Automating the hw mapping of dnn models on accelerators via genetic algorithm," in *Proceedings of the 39th International Conference on Computer-Aided Design (ICCAD)*, 2020.

[16] G. E. Moon, H. Kwon, G. Jeong, P. Chatarasi, S. Rajamanickam, and T. Krishna, "Evaluating spatial accelerator architectures with tiled matrix-matrix multiplication," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 2021.

[17] A. Symons, L. Mei, and M. Verhelst, "Loma: Fast auto-scheduling on dnn accelerators through loop-order-based memory allocation," in *2021 IEEE 3rd International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, 2021, pp. 1–4.

[18] Y. S. Shao, J. Clemons, R. Venkatesan, B. Zimmer, M. Fojtik, N. Jiang, B. Keller, A. Klinefelter, N. Pinckney, P. Raina *et al.*, "Simba: Scaling deep-learning inference with multi-chip-module-based architecture," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 14–27.

[19] Y.-H. Chen, T.-J. Yang, J. Emer, and V. Sze, "Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 2, pp. 292–308, 2019.

[20] M. Pellauer, Y. S. Shao, J. Clemons, N. Crago, K. Hegde, R. Venkatesan, S. W. Keckler, C. W. Fletcher, and J. Emer, "Buffets: An efficient and composable storage idiom for explicit decoupled data orchestration," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 137–151.

[21] C. Lattner, M. Amini, U. Bondhugula, A. Cohen, A. Davis, J. Pienaar, R. Riddle, T. Shpeisman, N. Vasilache, and O. Zinenko, "Mlir: Scaling compiler infrastructure for domain specific computation," in *CGO*, 2021.

[22] E. Mutlu, R. Tian, B. Ren, S. Krishnamoorthy, R. Gioiosa, J. Pienaar, and G. Kestor, "Comet: A domain-specific compilation of high-performance computational chemistry," in *Workshop on Languages and Compilers for Parallel Computing (LCPC'20)*. Springer.

[23] J. Wang, L. Guo, and J. Cong, "Autosa: A polyhedral compiler for high-performance systolic arrays on fpga," in *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '21, 2021, p. 93–104.

[24] L. Mei, P. Houshmand, V. Jain, S. Giraldo, and M. Verhelst, "Zigzag: Enlarging joint architecture-mapping design space exploration for dnn accelerators," *IEEE Transactions on Computers*, vol. 70, no. 8, pp. 1160–1174, 2021.

[25] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, M. Cowan, H. Shen, L. Wang, Y. Hu, L. Ceze, C. Guestrin, and A. Krishnamurthy, "Tvm: An automated end-to-end optimizing compiler for deep learning," in *Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'18, 2018, p. 579–594.

[26] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, "cudnn: Efficient primitives for deep learning," *arXiv preprint arXiv:1410.0759*, 2014.

[27] K. Raghavachari, G. Trucks, J. A. Pople, and M. Head-Gordon, "A fifth-order perturbation comparison of electron correlation theories," *Chemical Physics Letters*, vol. 157, pp. 479–483, 05 1989.

[28] E. Aprà, E. J. Bylaska, W. A. de Jong, N. Govind, K. Kowalski, T. P. Straatsma, M. Valiev, H. J. J. van Dam, Y. Alexeev, J. Anchell, V. Anisimov, F. W. Aquino, R. Atta-Fynn, J. Autschbach, N. P. Bauman, J. C. Becca, D. E. Bernholdt, K. Bhaskaran-Nair, S. Bogatko, P. Borowski, J. Boschen, J. Brabec, A. Bruner, E. Cauët, Y. Chen, G. N. Chuev, C. J. Cramer, J. Daily, M. J. O. Deegan, T. H. D. Jr., M. Dupuis, K. G. Dyall, G. I. Fann, S. A. Fischer, A. Fonari, H. Früchtl, L. Gagliardi, J. Garza, N. Gawande, S. Ghosh, K. Glaesemann, A. W. Götz, J. Hammond, V. Helms, E. D. Hermes, K. Hirao, S. Hirata, M. Jacquelin, L. Jensen, B. G. Johnson, H. Jónsson, R. A. Kendall, M. Klemm, R. Kobayashi, V. Konkov, S. Krishnamoorthy, M. Krishnan, Z. Lin, R. D. Lins, R. J. Littlefield, A. J. Logsdail, K. Lopata, W. Ma, A. V. Marenich, J. M. del Campo, D. Mejia-Rodriguez, J. E. Moore, J. M. Mullin, T. Nakajima, D. R. Nascimento, J. A. Nichols, P. J. Nichols, J. Nieplocha, A. O. de la Roza, B. Palmer, A. Panyala, T. Pirojsirikul, B. Peng, R. Peverati, J. Pittner, L. Pollack, R. M. Richard, P. Sadayappan, G. C. Schatz, W. A. Shelton, D. W. Silverstein, D. M. A. Smith, T. A. Soares, D. Song, M. Swart, H. L. Taylor, G. S. Thomas, V. Tipparaju, D. G. Truhlar, K. Tsemekhman, T. V. Voorhis, A. Vázquez-Mayagoitia, P. Verma, O. Villa, A. Vishnu, K. D. Vogiatzis, D. Wang, J. H. Weare, M. J. Williamson, T. L. Windus, K. Woliński, A. T. Wong, Q. Wu, C. Yang, Q. Yu, M. Zacharias, Z. Zhang, Y. Zhao, and R. J. Harrison, "NWChem: Past, Present, and Future," *Journal of Chemical Physics*, vol. 152, no. 17, p. 184102, May 2020.

[29] K. Kowalski, R. Bair, N. P. Bauman, J. S. Boschen, E. J. Bylaska, J. Daily, W. A. de Jong, D. T. H. Jr., N. Govind, R. J. Harrison, M. Keçeli, K. Keipert, S. Krishnamoorthy, S. Kumar, E. Mutlu,

B. Palmer, A. Panyala, B. Peng, R. M. Richard, T. P. Straatsma, P. Sushko, E. F. Valeev, M. Valiev, H. J. J. van Dam, J. M. Waldrop, D. B. Williams-Young, C. Yang, M. Zalewski, and T. L. Windus, "From NWChem to NWChemEx: Evolving with the computational chemistry landscape," *Chemical Reviews*, accepted for publication.

[30] "Nvidia tesla v100 gpu architecture." 2017, https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf.

[31] E. Qin, A. Samajdar, H. Kwon, V. Nadella, S. Srinivasan, D. Das, B. Kaul, and T. Krishna, "Sigma: A sparse and irregular gemm accelerator with flexible interconnects for dnn training," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2020, pp. 58–70.

[32] M. Gao, J. Pu, X. Yang, M. Horowitz, and C. Kozyrakis, "Tetris: Scalable and efficient neural network acceleration with 3d memory," in *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, 2017, pp. 751–764.

[33] K. Hegde, P.-A. Tsai, S. Huang, V. Chandra, A. Parashar, and C. W. Fletcher, "Mind mappings: Enabling efficient algorithm-accelerator mapping space search," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2021.

[34] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "How to evaluate deep neural network processors: Tops/w (alone) considered harmful," *IEEE Solid-State Circuits Magazine*, vol. 12, no. 3, pp. 28–41, 2020.

[35] "IREE: Intermediate representation execution environment," 2021, https://github.com/google/iree.

[36] "NPComp - mlir based compiler toolkit for numerical python programs," 2021, https://github.com/llvm/mlir-npcomp.

[37] "Tensor operator set architecture (tosa) dialect," 2021, https://mlir.llvm.org/docs/Dialects/TOSA/.

[38] R. Tian, L. Guo, J. Li, B. Ren, and G. Kestor, "A high-performance sparse tensor algebra compiler in Multi-Level IR," *arXiv preprint arXiv:2102.05187*, 2021.

[39] M. Naumov, D. Mudigere, H. M. Shi, J. Huang, N. Sundaraman, J. Park, X. Wang, U. Gupta, C. Wu, A. G. Azzolini, D. Dzhulgakov, A. Mallevich, I. Cherniavskii, Y. Lu, R. Krishnamoorthi, A. Yu, V. Kondratenko, S. Pereira, X. Chen, W. Chen, V. Rao, B. Jia, L. Xiong, and M. Smelyanskiy, "Deep learning recommendation model for personalization and recommendation systems," *CoRR*, vol. abs/1906.00091, 2019. [Online]. Available: http://arxiv.org/abs/1906.00091

[40] P. Springer and P. Bientinesi, "Design of a high-performance gemm-like tensor–tensor multiplication," *ACM Trans. Math. Softw.*, vol. 44, no. 3, 2018.

[41] Y. N. Wu, J. S. Emer, and V. Sze, "Accelergy: An architecture-level energy estimation methodology for accelerator designs," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2019, pp. 1–8.