SAND2021-7479R

Evaluating Spatial Accelerator Architectures with Tiled Matrix-Matrix Multiplication

Gordon E. Moon*, Hyoukjun Kwon†, Geonhwa Jeong†, Prasanth Chatarasi†, Sivasankaran Rajamanickam ‡, and Tushar Krishna§

Abstract—There is a growing interest in custom spatial accelerators for machine learning applications. These accelerators employ a spatial array of processing elements (PEs) interacting via custom buffer hierarchies and networks-on-chip. The efficiency of these accelerators comes from employing optimized dataflow (i.e., spatial/temporal partitioning of data across the PEs and fine-grained scheduling) strategies to optimize data reuse. The focus of this work is to evaluate these accelerator architectures using a tiled general matrix-matrix multiplication (GEMM) kernel. To do so, we develop a framework that finds optimized mappings (dataflow and tile sizes) for a tiled GEMM for a given spatial accelerator and workload combination, leveraging an analytical cost model for runtime and energy. Our evaluations over five spatial accelerators demonstrate that the tiled GEMM mappings systematically generated by our framework achieve high performance on various GEMM workloads and accelerators.

Index Terms—spatial accelerator, DNN accelerator, dataflow, GEMM mapping.

1 Introduction

Several custom ASIC accelerators have emerged in the recent past and have been successfully used to exploit massive parallelism and locality in the machine learning (ML) applications. The most popular examples are systolic arrays such as TPU [1], xDNN [2], RAPID [3] and advanced forms such as NVDLA [4], Eyeriss [5], ShiDianNao [6] and MAERI [7]. Accelerators such as these are already being integrated in Petascale systems. For example, the Lassen system at Lawrence Livermore National Labs has a Cerebras accelerator integrated with it. Recent results show a 0.86 PFLOPS on a single wafer scale chip [8] on stencil problems. Graphcore IPUs and SambaNova are starting to be use in traditional HPC applications [9], [10]. The advantages provided by these accelerators vary from reduced data movement due to the data flow on chip and opportunities to accelerate new applications that were not amenable to accelerators like GPUs. These benefits and recent successes demonstrate potential for these accelerators to be part of future exascale systems.

As more such heterogeneous systems are expected in the exascale era, it is important to develop a methodology for modeling such accelerators. 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 primary architectural features that distinguish these "spatial" accelerators for ML from CPUs and GPUs are parallelism using hundreds to thousands of processing elements (**PEs**), a fast network-on-chip (**NoC**) connecting these and use of private/shared scratchpad buffers for data reuse. Different accelerators differ in their **dataflow**, i.e., mechanism for data reuse over buffers and wires.

GEMM (General Matrix-Matrix Multiplication) is a key computational kernel within ML, computational science and engineering (CSE) applications, and other computational kernels [11], [12]. In this work, we focus on modeling and evaluating multiple spatial accelerator architectures using this important kernel. GEMM kernel has been studied heavily on traditional architectures in the context of CSE and ML workloads. Evaluating all the algorithm choices for GEMM, while considering workload dimensions, tiling strategies etc. on even a single new spatial accelerator is a challenge. We aim to do this evaluation on five different accelerators (via simulation). Optimizing the GEMM mapping on spatial accelerators poses the following challenges. (i) the sizes of input matrices vary significantly; (ii) the space of possible mappings involving multi-level tiling, parallelization, and loop orders could be in the billions (see Section 5.2 for an example); (iii) the dataflow choice of the accelerator and its internal structures (e.g., multi-cast, reduction tree) have to be accounted for as constraints (e.g., Table 2 summarizes these restrictions in five ML accelerators); (iv) accurate cost models to model diverse accelerator dataflows and estimate performance of a mapping on the accelerator are needed.

To address the challenges highlighted above and evaluate the different accelerators, we take a systematic approach. First, we develop a *methodology for modeling accelerators* for the tiled GEMM kernel (Section 3) which could be extended to other kernels in the future. Next, we develop a framework shown in Fig. 1 which is the *second contribution* of this work. Our framework finds the best mappings and tile sizes (defined based on projected runtime) of tiled GEMM over *multiple* spatial accelerators using an accurate analytical model. Our framework is comprised of two key components. (1) a mapping explorer **FLASH** (Flexible Linear Algebra dataflow via Spatio-temporal Hierarchical-mapping) that can explore a high dimensional space of tile sizes, loop ordering choices, parallelization, by pruning the search space

- * *Department of Software, Korea Aerospace University, Gyeonggi, Goyang, Republic of Korea (ehmoon@kau.ac.kr),*
- † *School of Computer Science, Georgia Institute of Technology, Atlanta, GA, 30332 (hyoukjun, geonhwa.jeong, cprasanth @gatech.edu).*
- ‡ *Center for Computing Research, Sandia National Laboratories, Albuquerque, NM, 87123 (srajama@sandia.gov). Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & 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-NA0003525.*
- § *School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA, 30332 (tushar@ece.gatech.edu).*

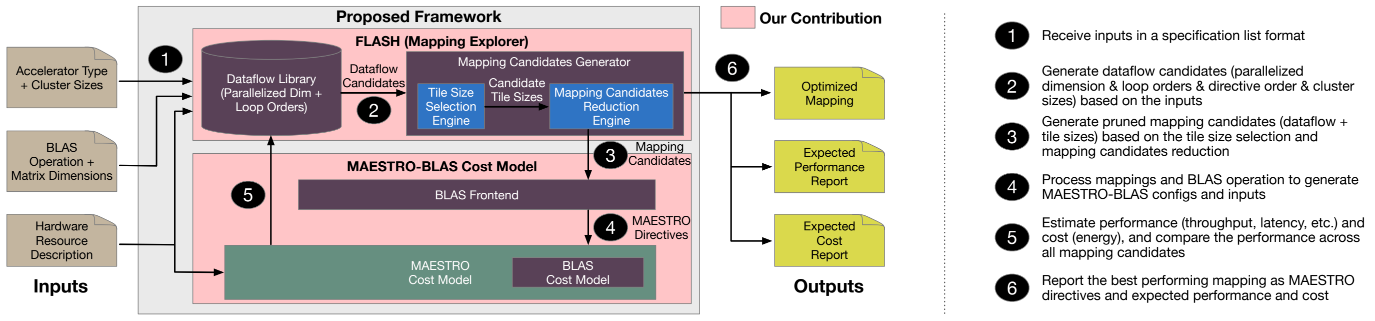

Proposed Framework
FLASH (Mapping Explorer)
Dataflow Library (Parallelized Dim + Loop Orders)
Dataflow Candidates
Mapping Candidates Generator
Tile Size Selection Engine
Candidate Tile Sizes
Mapping Candidates Reduction Engine
Mapping Candidates
MAESTRO-BLAS Cost Model
BLAS Frontend
MAESTRO Directives
MAESTRO Cost Model
BLAS Cost Model
Accelerator Type + Cluster Sizes
BLAS Operation + Matrix Dimensions
Hardware Resource Description
Inputs
Our Contribution
Optimized Mapping
Expected Performance Report
Expected Cost Report
Outputs
1 Receive inputs in a specification list format
2 Generate dataflow candidates (parallelized dimension & loop orders & directive order & cluster sizes) based on the inputs
3 Generate pruned mapping candidates (dataflow + tile sizes) based on the tile size selection and mapping candidates reduction
4 Process mappings and BLAS operation to generate MAESTRO-BLAS configs and inputs
5 Estimate performance (throughput, latency, etc.) and cost (energy), and compare the performance across all mapping candidates
6 Report the best performing mapping as MAESTRO directives and expected performance and cost

Fig. 1: An overview of the proposed framework for searching for optimal mappings with walk-through of steps.

given a workload, hardware description and dataflow constraints (addressing challenges (i)-(iii) above); (2) an analytical cost model **MAESTRO-BLAS** that can be used to evaluate the accelerators for mappings generated by FLASH (addressing challenge (iv)).

The *third contribution* of this work is a thorough experimental evaluation of the dataflows and tile sizes for six representative GEMM workloads on five popular spatial accelerators (which offer varying levels of flexibility in their dataflow) and two accelerator configurations (cloud and edge). We also evaluate the performance of the five accelerators on Deep Neural Networks (DNNs) where GEMM kernel is foundational to training and inference.

These contributions result in significant new results. The novel pruning approach in our framework reduces the search space by 99.7% for a $(256\times256)\times(256\times256)$ GEMM and still finds a correct mapping. This also leads to decreasing the time for searching through the space using MAESTRO-BLAS by 99.9%. We derive the tile sizes for GEMM on spatial accelerators analytically and show the tile sizes chosen by FLASH reduces runtime up to 94% and energy by up to 96%. The experimental evaluation using the analytical model shows the importance of loop order, matrix shape, and how flexible dataflow accelerators (e.g., MAERI [7]), can provide significant runtime and energy improvements as opposed to fixed dataflow accelerators. The novel coupling of search space pruning that accounts for hardware, workload, and algorithm choice with an analytical model is a generalizable framework for other workloads and architectures in the future.

2 Background

2.1 General Matrix-Matrix Multiplication (GEMM)

GEMM kernel multiplies two input matrices A of size $M\times K$ and B of size $K\times N$ to obtain an output matrix C of size $M\times N$, as shown in Algorithm 1. GEMM performs $M\times N\times K$ MACs (Multiply-ACcumulate operations) within three nested loops. The order of three loops can be changed to exploit various data reuses of matrices A, B and C across space and time.

Algorithm 1: General Matrix-Matrix Multiplication

```
Input: $A[M][K]$, $B[K][N]$
Output: $C[M][N]$
1 for $m = 0$ to $M-1$ do
2     for $n = 0$ to $N-1$ do
3         for $k = 0$ to $K-1$ do
4             $C[m][n] \mathrel{+}= A[m][k] \times B[k][n]$
```

Several state-of-the-art Deep Neural Network models spend a large fraction of the training/inference time on GEMM operations [13]. In addition, GEMM is the key computational kernel in CSE applications, level 3 BLAS kernels such as LU factorization, triangular matrix-matrix multiply, triangular-solve [11], [12], sparse direct methods for LU/QR factorizations [14], and sparse iterative solvers [15]. There are also smaller batched GEMM kernels that are critical for multiphysics codes [16], [17], [18]. Thus, addressing the performance of GEMM kernel would have a broad impact across CSE and ML applications. The primary difference between all these use cases is the size and shape of input matrices for GEMM. Experiments in this paper vary the size and shape of matrices, tile sizes and loop order to cover all these use cases. *The key difference that emerges in this study is that while one loop order is typically the best among GPUs (or CPUs) even from different vendors, the same is not true for spatial accelerators as they are quite diverse.*

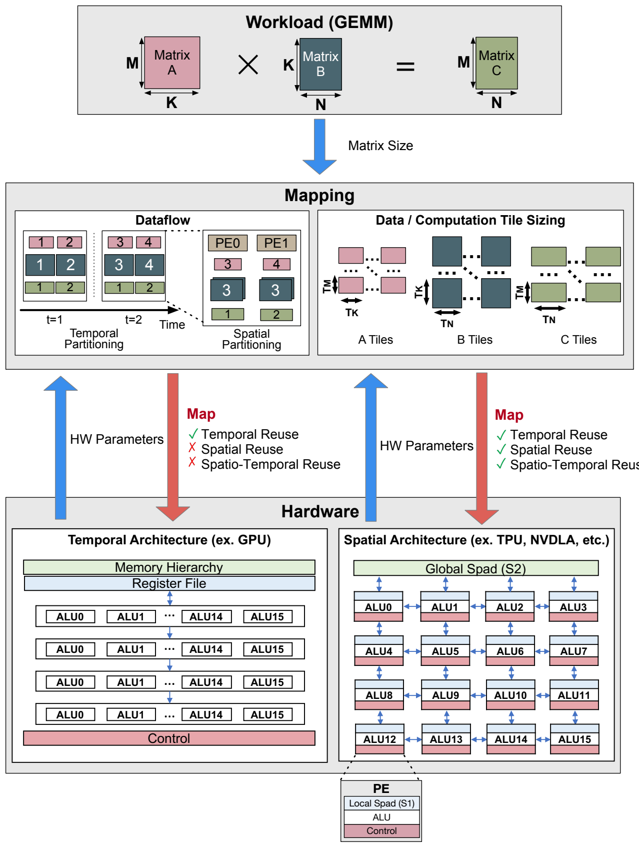


Fig. 2: An overview of mapping a GEMM workload on temporal architecture such as GPU, and spatial architecture such as TPU and NVLDA. GEMM mapping (middle) shows how the data for GEMM (top) could be partitioned (middle left) temporally or spatially and appropriate tile size are chose (middle right).

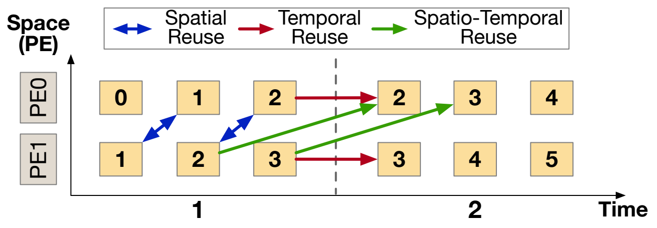


Fig. 3: Data reuse in the example mapping on two PEs. Spatial and spatial-temporal reuse are leveraged via the NoC wires while temporal reuse is leveraged via scratchpad buffers. Yellow boxes and numbers inside them represent data and data indices, respectively.

2.2 Spatial Accelerators

Architecture. Fig. 2 shows an abstract spatial accelerator architecture, which consists of a processing element (PE) array, a global shared scratchpad memory (called S2), and a controller. This template is common across all state-of-the-art ASIC accelerators [1], [4], [5], [6], [7]. Table 1 lists the ones we consider in this work. Network-on-chips (NoCs) interconnect the PEs and S2 global scratchpad memory. PEs are the compute units of accelerators that contain ALUs or Multiply-Accumulate units with a small local scratchpad (called S1). Like a cache memory, scratchpad memory reduces the number of remote buffer accesses. However, unlike a cache memory, data layout and insertion/eviction of data to a scratchpad memory are fully customizable for programmers or accelerator designers. Spatial accelerators exploit not only global buffer-PE communication but also inter-PE communication patterns [19] (i.e., dataflow among PEs and global buffer). The inter-PE communication pattern could be flexible or fixed. Fixed inter-PE communication pattern restricts which (GEMM) algorithm could be run on an accelerator. *Our framework takes into account the fixed/flexible communication patterns in addition to the typical architecture features such as number of PEs and S1/S2 size to faithfully simulate different accelerators.*

Data Reuse in Spatial Accelerators. Efficiency in spatial accelerators comes from exploiting data reuse. Data reuse in spatial accelerators can be categorized into three types using the temporal and spatial nature [20]. Fig. 3 provides an example mapping and data reuse analysis on the mapping. ***Temporal reuse*** occurs when a PE accesses the same set of data across time, and can be leveraged via the S1 scratchpad. For example, data 2 in Fig. 3 is accessed in PE0 across time 1 and 2 (red arrows). ***Spatial reuse*** occurs when multiple PEs access the same set of data at the same time via a multicast/broadcast over the NoC. For example, data 1 and 2 at time 1 in Fig. 3 are accessed by both PE0 and PE1 (blue arrows). ***Spatio-temporal reuse*** occurs when two adjacent PEs access the same set of data in a skewed schedule, and they are interconnected via point-to-point connections (i.e., wire). For example, data 2 and 3 in Fig. 3 are accessed in PE1 at time 1 and in PE0 at time 2. Those data can be directly forwarded from PE1 to PE0 so that PE0 does not need to fetch the data from S2. The ability of an accelerator to leverage all three kinds of reuse depends on its internal microarchitecture (i.e., size of S1 and S2 buffers for temporal and NoC topology for spatial/spatio-temporal). *These reuse features have to be taken into account in addition to scratchpad sizes when developing mapping strategies for any algorithm running on this hardware.*

Comparison to GPUs. Fig. 2 also shows the differences between spatial accelerators and temporal ones such as GPUs. Some of the major logical differences are 1) ALUs in spatial accelerators communicate directly using the NoC without register file intervention, unlike ALUs in GPU's which communicate via writing and reading from the register file, and 2) Spatial accelerators provide spatial, temporal, and spatio-temporal data reuse (via hardware support for multi-cast, broadcast, direct forwarding and spatial reduction as part of the NoC) as opposed to only temporal reuse (via scratchpads) on GPUs. This allows spatial accelerators to achieve higher performance, better runtime and lower energy usage.

To summarize, spatial accelerators are considerably different from traditional accelerators and diverse. Developing a framework (Section 4) to map widely used kernels on these accelerators and evaluating these accelerator designs (Section 5) is a key first step in understanding their value. This is the focus of this work.

2.3 Dataflow Directives and Mapping

Dataflow. Spatial accelerators carefully encode the data movement for reuse and parallelization into the accelerator's hardware microarchitecture. This is also known as "**dataflow**"[1]. Specifically, the dataflow includes two key components: *(i) parallelization strategy*, i.e., which dimensions of the tensors can be run in parallel. *(ii) computation order*, i.e., the order in which the dimensions of the tensor are scheduled over the accelerator.

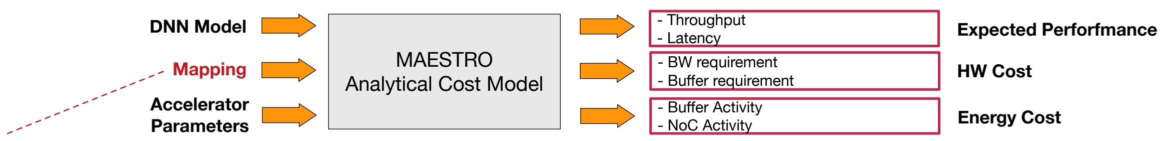
The dataflow has two implications. First, it determines the amount of data reuse on the inputs and outputs. Prior work has studied this relationship for the ML use case [20], [21] and concluded that the reuse efficiency of the dataflow depends on the target DNN layer dimensions and shape. *We extend this to GEMM of different sizes and shapes by developing a framework that can find the optimal tile sizes and pruning the search space for a variety of workloads and accelerator combinations.* Second, dataflow choices also have implications on accelerator microarchitecture aspects. The parallelization strategy directly affects the NoC implementation. Also, given accelerator microarchitecture (as in this study), the dataflow choices are restricted. For example, parallelism across the dimension being reduced (e.g., K in a GEMM or input channels in a convolution) leads to different PEs computing partial sums for the same output and needs reduction via the NoC (e.g., a reduction chain or tree). *We take these hardware functionality into account when mapping GEMM to an accelerator.* Supporting complex dataflows requires more complexity in the accelerator hardware [7], [20]. Different spatial accelerators (commercial [1], [4] and research prototypes [5], [6]) have picked different dataflows [21] trading off reuse-efficiency and hardware complexity.

Dataflow Directives. In this work, we leverage the *dataflow directives* introduced by MAESTRO [20] to express the exact dataflows for various accelerators under consideration.

There are three directives: TemporalMap, SpatialMap, and Cluster which are described in Fig. 4. TemporalMap implies that the data changes over time, and remains same over space (i.e., across PEs). SpatialMap implies that the data changes over space (i.e., parallelism). Cluster helps describe hierarchical dataflows by grouping PEs into clusters of certain Size.

One could recursively define dataflows within clusters. For example, NVDLA [4] maps convolutions by employing SpatialMap across input channels as its intra-cluster dataflow, and SpatialMap across output channels as its inter-cluster dataflow. The relative

1. Note that this is different from dataflow style of programming that was popular few decades ago.



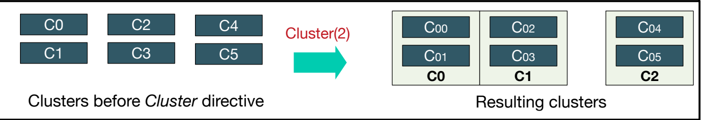
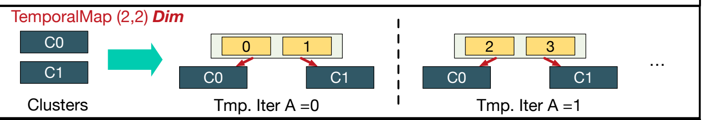
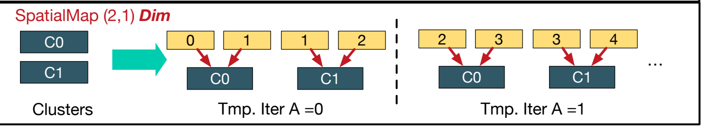
Syntax	Description	Example
Cluster (*Size*)	Create a group of ***Size*** number of PEs. Used to create hierarchies (e.g., 2D PE array can be viewed as a cluster of 1D PE clusters). SpatialMap and TemporalMap can be applied across clusters or PEs.	
TemporalMap (*Size, Offset*) *Dim*	Map ***Size*** number of data indices of ***Dim*** on a cluster (or PE). Data indices change over time with increments of ***Offset*** and remain the same across space (i.e., PEs).	
SpatialMap (*Size, Offset*) *Dim*	Map ***Size*** number of data indices of ***Dim*** on a cluster (or PE). Data changes over space (i.e., PEs) with increments of ***Offset***. They may remain the same or change across time depending on the ordering of directives.	

Fig. 4: Syntax and description of MAESTRO dataflow directives.

order among the directives specifies computation order. We provide a concrete example later in Section 3.2. The specific amount of reuse depends on the overall dataflow, workload dimensions and tile sizes (together called mapping).

TABLE 1: Specifications of target spatial accelerator architectures. Note that *input-*, *weight-*, and *output-stationary* dataflows correspond to input matrix A-, input matrix B-, and output matrix C-stationary dataflows in GEMM operation, respectively. More details of the dataflow are in Table 2. For our evaluations, we provide equal hardware resources (PEs and buffers) to all accelerators (Table 4).

Spatial Accelerators	HW Configuration		Dataflow
	# PEs (row×col)	NoC	
Eyeriss [5]	12×14	Buses	*input row stationary*
NVDLA [4]	64×8	Bus + Tree	*weight-stationary*
TPUv2 [1]	128×128	Mesh	*weight-stationary*
ShiDianNao [6]	8×8	Mesh	*output-stationary*
MAERI [7]	256 (any aspect ratio)	Custom Fat Tree	*flexible*

Mapping. The number of compute units and amount of on-chip buffers in accelerators are typically not sufficient to cover entire workload (e.g., BLAS operations on a node are on matrices of size $O(10^3 - 10^4)$ and Resnet50 [22] contains 3.8×10^9 FLOPs while accelerators have hundreds [23] to thousands [1] PEs). Therefore, tiling the computation and data of workload and time-multiplexing the target accelerator is necessary.

The relationship between mapping and dataflow is shown in Fig. 2. Specifically, the dataflow of the accelerator, the tile sizes for all tensors, and scheduling of these tiles for the entire workload is known as a **mapping**, as shown in Fig. 2. The mapping determines the data needed at each PE at each instance of time. The proportion of this data that needs to be moved across the memory hierarchy (from off-chip DRAM to S2, and from S2 to S1) and the proportion that can be reused depends on the accelerator's dataflow.

An optimized mapping is crucial for overall efficiency, since data movement dominates energy in accelerators [23]. The Size parameter in the MAESTRO's SpatialMap and TemporalMap directives in Fig. 4 can be used to specify the tile sizes for each dimension of the matrices.

3 Accelerator Modeling Methodology

Table 1 shows the key features of five target spatial accelerators explored in our study. The diversity of the chosen accelerators adds to the complexity of evaluating even one kernel on all of them. We discuss our methodology below.

3.1 Modeling Spatial Accelerators using Dataflow Directives

A key challenge in performing an apples-to-apples comparison of the accelerators described above is the fact that they differ in the number of PEs, buffer sizes, dataflow, internal microarchitecture (and software stacks [1], [4] or lack-there-of for research prototypes [5], [6], [7]). To address this issue, we contrast the accelerators based on "how" they map GEMM on the spatial substrate[2]. Furthermore, in our evaluations, we provide the same hardware resources to all accelerators (Table 4)[3].

Example of GEMM Mapping. We employ the dataflow directives described in Section 2.3 to analyze and describe how each accelerator runs a GEMM, constrained by their dataflow. We refer to the GEMM mappings using the abbreviated directive order, where we use **T** for TemporalMap, **S** for SpatialMap and _ for introducing a cluster (i.e., hierarchy). For example, a simple **TTT** mapping is nothing but triple nested GEMM loop on sequential core with the loop order determined by the binding of M, N and K loops (Section 2.1) to each directive; **TTT_TTT** is the hierarchical/cache-blocked version of the same. An **STT** mapping with MNK as the loop order would be a triple nested GEMM where rows of matrix A (i.e., dimension M) are spatially mapped (i.e., parallelized) across CPU cores or GPU CUDA threads or PEs in a spatial accelerator.

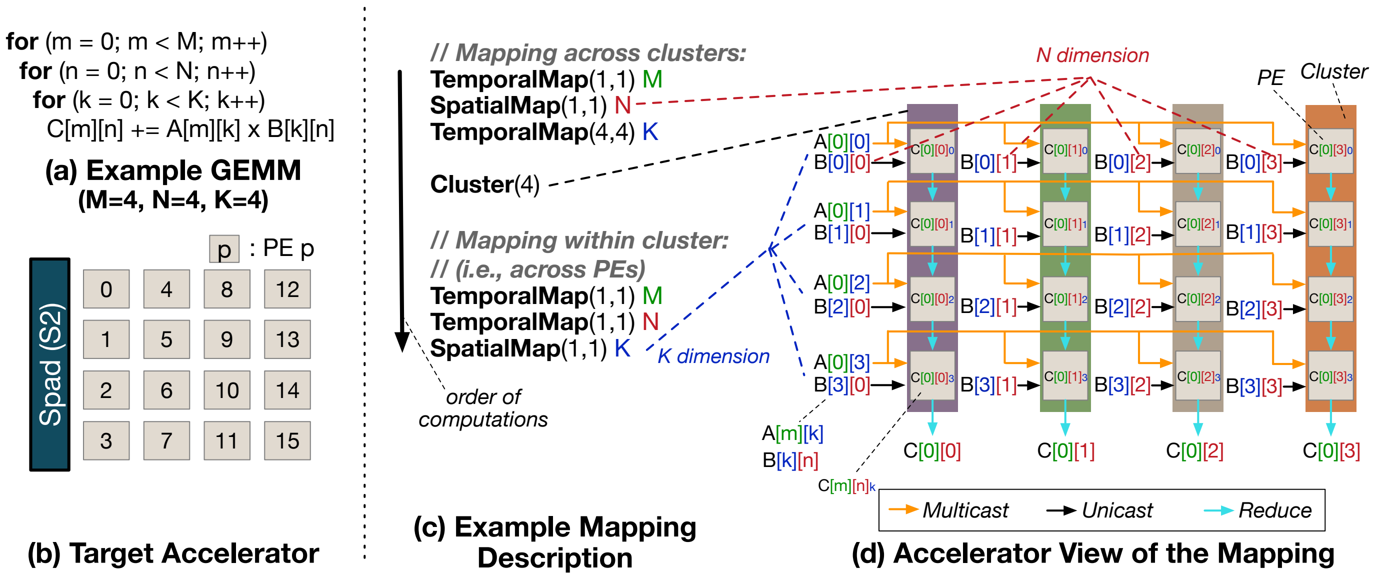
GEMM Mappings supported by Spatial Accelerators. Table 2 presents the GEMM mappings that can be supported by our target accelerators. Each mapping has directives for describing the dataflows both across the clusters (written above the Cluster

2. Note that except for the TPU, the other four accelerators are originally designed to directly run convolutions and leverage reuse across sliding windows. We map GEMM on these convolution accelerators by expressing it as a convolution with one row and one channel. To the best of our knowledge, previous studies at mapping GEMM have not addressed how to map GEMM efficiently on existing state-of-the-art spatial accelerators.

3. This is why we refer to our mapping descriptions as *-style as they are not running on the actual accelerator instances.

TABLE 2: Comparison of various GEMM mappings constrained by state-of-the-art spatial accelerators where P and λ denote total number of PEs and size of cluster (i.e., number of PEs in each cluster), respectively. $T^{out/in}_{dim.}$ denotes the tile size for each of dimensions M, N and K in the outer/inner clusters. TMap and SMap refer to TemporalMap and SpatialMap directives, respectively.

Spatial Accelerator	Eyeriss [5]	NVDLA [4]	TPU [1]	ShiDianNao [6]	MAERI [7]
Dataflow: Parallel Dim	Inter-Cluster: M Intra-Cluster: K	Inter-Cluster: N Intra-Cluster: K	Inter-Cluster: N Intra-Cluster: K	Inter-Cluster: M Intra-Cluster: N	Inter-Cluster: M or N or K Intra-Cluster: M or N or K
Dataflow: Compute Order	Inter-Cluster: $<m, n, k>$ Intra-Cluster: $<m, n, k>$	Inter-Cluster: $<n, k, m>$ Intra-Cluster: $<n, m, k>$	Inter-Cluster: $<n, m, k>$ Intra-Cluster: $<n, m, k>$	Inter-Cluster: $<m, n, k>$ Intra-Cluster: $<m, n, k>$	Inter-Cl.: $<m/n/k, n/m/k, k/m/n>$ Intra-Cl.: $<m/n/k, n/m/k, k/m/n>$
Cluster Size (λ)	$1 \leq \lambda \leq 12$ (compile time-flexible)	$16 \leq \lambda \leq 64$ (design time-flexible)	256 or $\sqrt{P}$	8 or $\sqrt{P}$	$T^{out}_M / T^{out}_N / T^{out}_K$ (tile size of the last dimension)
GEMM Mapping	**SMap** (T^{out}_M, T^{out}_M) M **TMap** (T^{out}_N, T^{out}_N) N **TMap** $(T^{out}_K \times \lambda, T^{out}_K \times \lambda)$ K **Cluster** (λ) **TMap** (T^{in}_M, T^{in}_M) M **TMap** (T^{in}_N, T^{in}_N) N **SMap** (T^{out}_K, T^{out}_K) K	**SMap** (T^{out}_N, T^{out}_N) N **TMap** $(T^{out}_K \times \lambda, T^{out}_K \times \lambda)$ K **TMap** (T^{out}_M, T^{out}_M) M **Cluster** (λ) **TMap** (T^{in}_N, T^{in}_N) N **TMap** (T^{in}_M, T^{in}_M) M **SMap** (T^{out}_K, T^{out}_K) K	**SMap** (T^{out}_N, T^{out}_N) N **TMap** (T^{out}_M, T^{out}_M) M **TMap** $(T^{out}_K \times \lambda, T^{out}_K \times \lambda)$ K **Cluster** (λ) **TMap** (T^{in}_N, T^{in}_N) N **TMap** (T^{in}_M, T^{in}_M) M **SMap** (T^{out}_K, T^{out}_K) K	**SMap** (T^{out}_M, T^{out}_M) M **TMap** $(T^{out}_N \times \lambda, T^{out}_N \times \lambda)$ N **TMap** (T^{out}_K, T^{out}_K) K **Cluster** (λ) **TMap** (T^{in}_M, T^{in}_M) M **SMap** (T^{out}_N, T^{out}_N) N **TMap** (T^{in}_K, T^{in}_K) K	**TMap** (T^{out}_M, T^{out}_M) M **SMap** (T^{out}_N, T^{out}_N) N **TMap** (T^{out}_K, T^{out}_K) K **Cluster** (T^{out}_K) **TMap** (T^{in}_M, T^{in}_M) M **TMap** (T^{in}_N, T^{in}_N) N **SMap** $(1,1)$ K
Mapping Name	STT_TTS-MNK (Eyeriss-style)	STT_TTS-NKM (NVDLA-style)	STT_TTS-NMK (TPU-style)	STT_TST-MNK (ShiDianNao-style)	TST_TTS-MNK (MAERI-style)


```
for (m = 0; m < M; m++)
  for (n = 0; n < N; n++)
    for (k = 0; k < K; k++)
      C[m][n] += A[m][k] x B[k][n]
```
```
// Mapping across clusters:
TemporalMap(1,1) M
SpatialMap(1,1) N
TemporalMap(4,4) K

Cluster(4)

// Mapping within cluster:
// (i.e., across PEs)
TemporalMap(1,1) M
TemporalMap(1,1) N
SpatialMap(1,1) K
```

Fig. 5: Walk-through example of TST_TTS-MNK (i.e., MAERI-style) GEMM mapping. (a) shows the example GEMM with for loops and (b) shows the abstract picture of the target accelerator in this example. (c) is the example mapping that is described using the aforementioned dataflow directives. (d) shows the assignment of entries of input matrices A and B to the PEs in four clusters. Each cluster computes one entry of output matrix C. We evaluate five such accelerators with different hardware configurations.

directive) and within the cluster (written below the Cluster directive). *This implies that all GEMM mappings use a 2D-tiled approach*. The cluster size is tied to accelerator microarchitecture (e.g., it represents the number of PE rows in Table 1), while the maximum tile size is constrained by S1 and S2 sizes.

Each mapping encodes the dataflow and hardware constraints from the specific accelerator. We provide some details next. As mentioned in Section 2.3, the dataflow determines the *parallelizing dimension*, and *compute order*.

We first focus on the parallelizing dimension. Recall that the K dimension in a GEMM is reduced when generating the output. Eyeriss, TPU, NVDLA and MAERI all provide NoC support for spatial reduction; Eyeriss and TPU via store-and-forward across the column and NVDLA and MAERI via reduction trees. Thus, for GEMM, we map the K dimension spatially within each cluster for these four accelerators. Outside the cluster, the same dimension needs to be mapped temporally, with a tile size that can cover the tiles needed by all the clusters. Either M or N can be mapped spatially outside the clusters. In ShiDianNao, however, there is no support for spatial reduction, and hence the K dimension is mapped temporally; the parallelism comes from N dimension instead within the cluster.

Next, the compute (or loop order) is determined by the relative order of the temporal directives. The TPU and NVDLA keep the weight matrix (i.e., matrix B) in GEMM stationary, and stream the input matrix (i.e., matrix A). This is reflected by keeping the N dimension as the outermost loop, both within and across the clusters. Eyeriss keeps the input rows stationary, and this is specified by keeping M as the outermost dimension. ShiDianNao keeps the output stationary, forcing a M followed by N loop order. MAERI allows all loop orders[4]. In each mapping, the tile sizes are free variables, constrained by the S1 buffer (for intra-cluster) and S2 buffer (for inter-cluster). Our policy for determining optimized tile sizes for GEMM mappings is described in Section 4.

4. Accelerators like LAP [24] implement GEMM with SUMMA algorithm. SUMMA is a subset of the MAERI-style TST_TTS mapping with the $<k, m, n>$/$<k, n, m>$ loop order.

3.2 Walk-through Example of TST_TTS Mapping

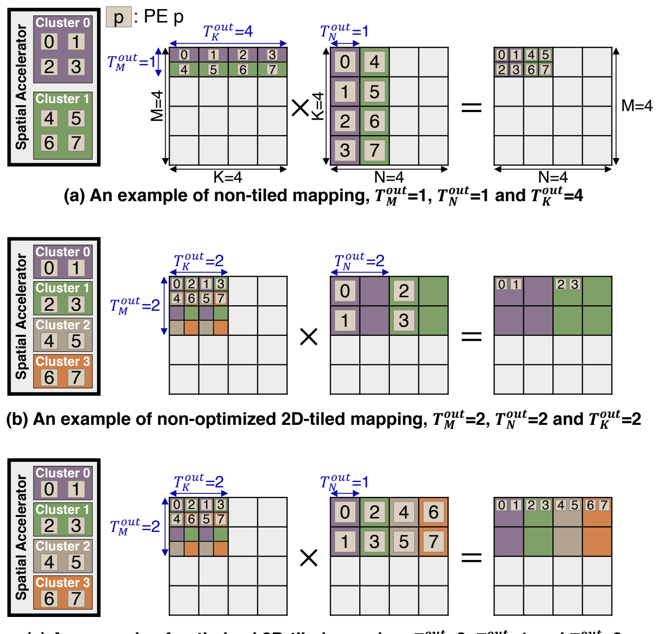
Fig. 5(a) shows a simple GEMM (with M=4, N=4 and K=4) which we want to map on a 16 PE spatial accelerator (Fig. 5(b)). Fig. 5(c) shows an example mapping for this GEMM using the dataflow directives. Note that this TST_TTS (MAERI-style) mapping is one of many possible mappings as we discussed in Section 3.1. We describe three key facets of this mapping, that can be seen in Fig. 5(d) from the perspective of the accelerator: Clustering, Intra-Cluster Behavior, and Inter-Cluster Behavior.

(i) *Clustering.* In this mapping, the 16 PEs are divided into four clusters, 4 PEs each (specified via the Cluster directive) corresponding to the accelerator columns in Fig. 5(d). This allows the mapping to exploit the 2D PE array by spatially distributing two GEMM dimensions.

(ii) *Intra-Cluster Behavior.* Directives below the cluster directive specify the intra-cluster mapping. Within each cluster (i.e., column of the accelerator in Fig. 5(d)), the K dimension is mapped spatially (specified via the SpatialMap), while the M and N dimensions are mapped temporally (i.e., remain same across all the PEs and only change with time). All directives use a *Size* and *Offset* of 1. These directives specify that each PE receives *one* unique element from the row and column of the matrices A and B respectively, since M and N stay the same, but K changes in each PE. This can be visualized from Fig. 5(d). During this operation, each PE computes a partial sum and forwards it to its neighbor along the column for accumulation. Each cluster thus computes one element of matrix C.

(iii) *Inter-Cluster Behavior.* Directives above the cluster directive specify the inter-cluster mapping. Across the clusters, the N dimension is mapped spatially, while the M and K dimensions are temporal. This means that the elements of the matrix B gets distributed across the different clusters, while the elements of the matrix A remain the same (and can thus be multicast). The *size* and *offset* field for K is 4 to specify that each cluster receives 4 elements from each matrix (which then get distributed among the 4 PEs within the cluster as discussed above). If this field is not set appropriately, it can lead to under-utilization within the cluster. FLASH takes care of this as we discuss later in Section 4. From Fig. 5(d), it can be seen that each time-step of the mapping computes one row of outputs $C_{0,:}$ for the matrix C, and would move on to the next row in the next time-step. If the dimensions of the matrix exceed the dimensions of the physical array, the computation would need to be tiled. The computation order across tiles of the three matrices depends on the order in which the directives are specified (Fig. 5(c)).

Example of Tiling in MAERI-style (TST_TTS) Mapping. Fig. 6 shows the impact of different tile sizes on the matrices A and B of size 4×4 with eight PEs. We use the notation T_M^{out} (T_N^{in}) for tile size of M (N) dimension in outer (inner) loop or inter-cluster (intra-cluster). Assume that T_M^{out}=T_M^{in}=1, T_N^{out}=T_N^{in}=1, T_K^{out}=4, and T_K^{in}=1 as shown in Fig. 6 (a). Suppose there are total of two clusters that contain four PEs each. The four PEs in each cluster are spatially mapped onto dimension K (column of A and row of B). Since two clusters are spatially mapped onto dimension N, entire eight PEs are fully utilized as PEs 0–3 and 4–7 can compute $C_{0,0}$ and $C_{0,1}$ entries, respectively. However, this non-tiled mapping does not provide the best performance. Hereafter, given any loop order, if the parallelism in the outer cluster is only on the innermost dimension (which is K in $<m, n, k>$ loop order) and the tile sizes of *both* outer dimensions (M and N here) are set



(a) An example of non-tiled mapping, T_M^{out}=1, T_N^{out}=1 and T_K^{out}=4

(b) An example of non-optimized 2D-tiled mapping, T_M^{out}=2, T_N^{out}=2 and T_K^{out}=2

(c) An example of optimized 2D-tiled mapping, T_M^{out}=2, T_N^{out}=1 and T_K^{out}=2

Fig. 6: Three examples of TST_TTS-MNK mapping when the tile sizes T_M^{out}, T_N^{out} and T_K^{out} are varied. For all examples, M=4, N=4, K=4, total number of PEs P=8, T_M^{in}=T_M^{out}, T_N^{in}=T_N^{out} and T_K^{in}=1.

to 1, we will call this **non-tiled mapping**. Fig. 6 (b) shows a 2D tiling case when T_M^{out}=T_M^{in}=2, T_N^{out}=T_N^{in}=2, T_K^{out}=2, and T_K^{in}=1. Such mappings with non-unit tile sizes are called **tiled mapping**. The tile size chosen T_N^{out}=2 yields under-utilization of clusters as only two out of four clusters can be mapped onto N dimension, i.e., the last two clusters, Clusters 2 and 3, will be idle. This results in under-utilization of PEs 4–7 in the accelerator. In the case shown in Fig. 6 (c), the tile size T_N^{out} is determined by $N/$(*number of clusters*). The number of clusters in MAERI-style (TST_TTS) mapping is associated with the total number of PEs and the tile size of the dimension involving the last loop (which is T_K^{out} in $<m, n, k>$ dataflow). Hence, the optimal tile size T_N^{out} can be calculated by T_N^{out}=$N/(P/T_K^{out})$ where P denotes total number of PEs. Fig. 6 (c) shows an example of 2D-tiled TST_TTS-MNK mapping when the appropriate tile sizes T_M^{out}=T_M^{in}=2, T_N^{out}=T_N^{in}=1, T_K^{out}=2, and T_K^{in}=1 are chosen for fully utilizing PEs and *maximizing data reuse*. We describe how we chose the tile sizes analytically in Section 4. The key observation is that Clusters 0–3 compute partial outputs of four entries in matrix C in parallel, while enabling data reuse of entries in matrix B. While the idea is similar to tiling on temporal architectures, the mix of spatial and temporal data reuse has to be handled when tiling.

3.3 MAESTRO-BLAS cost model

Given a GEMM mapping described via the dataflow directives, we evaluate its performance on a target spatial accelerator using our cost model which we call MAESTRO-BLAS. MAESTRO-BLAS builds upon a previously published open-source analytical cost model called MAESTRO [20], [25] MAESTRO receives a DNN model, accelerator hardware configuration, and mapping description as inputs, as shown in Fig. 4. It analytically (i.e., via detailed equations) analyzes the inputs and produces the expected runtime, number of buffer accesses, arithmetic intensity, NoC bandwidth requirement, and so on as outputs. It also reports the energy consumption based on energy of HW building blocks of accelerators (fixed point MAC units, SRAMs, buses, and so on) from CAD tools which are scaled based on the hardware configuration provided as input [20]. Its results have been validated against the Eyeriss chip [5] and RTL simulations of MAERI [7]. MAESTRO, however, does not accept BLAS operations as is, and requires users to map BLAS operations such as GEMM onto DNN operators (e.g., CONV2D). Our MAESTRO-BLAS updates adds native BLAS kernel frontend in MAESTRO, which allows users to directly work on BLAS operations without converting them into DNN operators. MAESTRO-BLAS leverages MAESTRO's backend for the performance and energy estimates.

4 FLASH: A Mapping Explorer for GEMM

Fig. 1 shows an overview of our framework FLASH for exploring GEMM mappings. We describe how we derive candidate tile sizes for the accelerators, prune the search space using dataflow, hardware and tile sizes constraints, and evaluate the mappings using the MAESTRO-BLAS cost model.

Given a GEMM dataflow for an accelerator-style, a GEMM workload and hardware configurations, exploring search space for all possible mappings is extremely compute-intensive if every feasible loop order, every cluster size, and every possible tile size are taken into consideration. Therefore, we propose a new search space reduction algorithm which systematically prunes the possible mappings by seeking the candidate (near optimal) tile sizes within the FLASH mapping explorer.

Dataflow and Hardware Constraints. FLASH honors the hardware constraints (expressed via the dataflow in Table 2) across the various accelerators, as described next. (i) The GEMM dimensions bound to the directives (i.e., loop order) is fixed except for MAERI-style mapping where all loop orders are supported by the hardware. (ii) Cluster size is fixed in every GEMM mapping except the MAERI-style mappping (TST_TTS). (iii) Tile size is a flexible input to any directive.

Candidate Mapping Selection. Algorithm 2 in Appendix shows the pseudo-code for generating mapping candidates in FLASH. Given the type of accelerators (e.g., Eyeriss/NVDLA), hardware parameters (e.g., S1/S2 buffer sizes), and the sizes of M, N, and K as inputs, FLASH generates mapping candidates. Based on the accelerator chosen by the user, FLASH first determines three parameters – dataflow directive order, all feasible loop orders, and all possible cluster sizes that can be explored in the mapping candidates based on the hardware constraints (line 1). Next, FLASH computes the candidate tile sizes using problem dimensions and hardware parameters (lines 7 and 8). These tile sizes are used to prune a huge number of possible mappings (lines 9 and 10).

Tile Size Selection. Given an arbitrary accelerator and a large-scale GEMM workload, for each feasible loop order, all possible combinations of the tile sizes can be obtained by using the maximum tile size acceptable for each dimension in outer and inner clusters. In most cases, each of the maximum tile sizes in the outer-cluster are equivalent to its actual dimension size *if there are no other constraints*. We consider these as the baseline candidates. Three constraints are imposed to reduce the candidates further: (1) the inter-cluster tile sizes of the three matrices, A, B and C, should fit within the S2 buffer; (2) the required memory for all clusters to be less than the S2 buffer; (3) the required memory for each PE to be less than the S1 buffer. In other words, the inner tile sizes must fit into the S1 buffer and must be a subset of the outer tile sizes (e.g., $T_M^{in} <= T_M^{out}$). Determining the candidate tile sizes is shown in lines 7–9 in Algorithm 2 in Appendix.

We show how these tile sizes can be calculated for one example, the MAERI-style TST_TTS mapping with <m, n, k> loop order (last column of Table 2) using the constraints discussed above.

For a MAERI-style mapping, note T_N^{out} can be computed as described in Section 3.2 because N is spatially-mapped. Hardware parameters such as S1/S2 buffer sizes must be considered to calculate near optimal candidate tile sizes T_M^{out} and T_K^{out} involved in the temporally-mapped dimensions M and K, respectively. Hence, given S2 and S1 buffer sizes, the tile sizes for outer cluster and inner cluster will be limited by the constraints in Equations 1 and 2, respectively. We assume double-buffering for latency hiding in these calculations.

$$T_M^{out} \times T_K^{out} + T_K^{out} \times \left(T_N^{out} \times \left(\frac{P}{T_K^{out}}\right)\right) + T_M^{out} \times \left(T_N^{out} \times \left(\frac{P}{T_K^{out}}\right)\right) \le \beta \times \frac{1}{2} \quad (1)$$

$$T_M^{in} \times T_K^{in} + T_K^{in} \times T_N^{in} + T_M^{in} \times T_N^{in} \le \alpha \times \frac{1}{2} \quad (2)$$

where α and β denote S1 buffer size and S2 buffer size, respectively. For the candidate outer tile sizes for S2 buffer size, as the dimension N is spatially mapped in the outer cluster-level, T_N^{out} in Equation 1 can be replaced with $N/(P/T_K^{out})$. Then if T_M^{out} and T_K^{out} are assumed to be equal, the candidate tile sizes are shown in Equation 3. We iteratively decrease the largest tile size when the tiles do not fit in the S2 buffer. Such corner cases might occur due to assumptions like T_K^{out} and T_M^{out} are the same.

$$1 \le T_M^{out} \le \sqrt{\frac{\beta}{2} + N^2} - N \ , \ 1 \le T_K^{out} \le \sqrt{\frac{\beta}{2} + N^2} - N \ , \ T_N^{out} = \frac{NT_K^{out}}{P} \quad (3)$$

If T_M^{in} and T_N^{in} are assumed to be equal, the solution to Equation 2 for S1 buffer size is shown in Equation 4 (the tile size T_K^{in} is always one in MAERI-style <m, n, k> loop order).

$$1 \le T_M^{in} \le \sqrt{\frac{\alpha+2}{2}} - 1 \ , \ 1 \le T_N^{in} \le \sqrt{\frac{\alpha+2}{2}} - 1 \ , \ T_K^{in} = 1 \quad (4)$$

For the tile sizes T_M^{out}, T_K^{out}, T_M^{in} and T_N^{in}, the largest power of two (constrained by Equations 3 and 4) result in better performance with respect to energy consumption and execution time. Equations 1 and 2 can be used to find the candidate tile size for different mapping schemes such as STT_TTS and STT_TST. We avoid derivations for other mappings due to space constraints. The candidate tile sizes for other mapping schemes can be derived in a similar way. Table 6 in Appendix summarizes the candidate tile sizes for all mapping schemes for all accelerators based on the same inequalities.

Mapping Candidates Generation. Once the candidate tile sizes are selected for each matrix dimension in the outer and inner clusters, all other tile sizes outside this range can be pruned. The tile sizes can then be combined with the chosen directive, loop order, cluster size selections to enumerate all the mapping candidates.

Mapping Candidates Selection using MAESTRO-BLAS. Given the pruned mapping candidates generated by Algorithm 2 in Appendix, FLASH uses them as inputs to the MAESTRO-BLAS cost model described earlier in Section 3.3 and selects the best mapping based on the lowest projected runtime.

5 Experimental Evaluation

In this section, we divide the evaluation into two parts. First we demonstrate the impact of our framework by showing the effect of pruning in Section 5.2 and tiling in Section 5.3. Second, we evaluate the accelerator architectures for different matrix shapes, loop orders and cluster sizes in Section 5.4.

5.1 Evaluation Methodology

GEMM Workloads. We select six distinct GEMM workloads, as listed in Table 3, inspired from several use cases to cover a wide variety of sizes and shapes [11], [14], [15]. The workload includes GEMM with large number of FLOPs (up to 549 GFLOPs) and different aspect ratios to show impact. We also choose the shapes considering their applicability in real applications especially, tall-skinny, short-fat, square, rank-K update like problems.

TABLE 3: The GEMM workloads we use for evaluations.

Matrix	Workload ID					
Dimension	I	II	III	IV	V	VI
M	8192	1024	8	8	8192	512
N	8192	1024	8	8192	8	256
K	8192	8192	8192	1024	1024	256
GFLOPs	549.8	8.59	0.001	0.067	0.067	0.03

Mappings. We use five mapping schemes following the style of the five accelerators, as shown in Table 2. As MAERI-style mapping TST_TTS has the flexibility to use different loop order, we use <m, n, k> loop order unless specified. We use the best performing candidate tile sizes computed using the method discussed in Section 4 or their closest power of two that fit in S2 buffer. We also select the best cluster size parameter (λ) for each mapping considering the micro-architectural constraints from accelerators that motivated each mapping (Table 2).

Hardware Configuration. For a fair evaluation of the design of the architecture (as opposed to the instance of an architecture), we use the same hardware parameters (number of PEs, buffer sizes and NoC bandwidth) for each one of them targeting edge and cloud devices with a 2D PE array, as described in Table 4. We assume 1GHz clock and 28nm node. The configurations are based on those of previously proposed DNN accelerators [1], [4], [5] to show the performance of the accelerators under realistic settings.

TABLE 4: Hardware configurations we use for evaluations. We assume a 1 GHz clock for the accelerators at 28nm. Performance goal is based on the number of PEs and the clock rate.

ID	# of PEs	S1 Size	S2 Size	NoC BW	Perf FLOPS	Off-chip Mem
Edge	256	0.5 KB	100 KB	32 GB/s	256 G	DRAM
Cloud	2048	0.5 KB	800 KB	256 GB/s	2 T	HBM

.

Hardware Modeling.. The modeling of each accelerator is done by MAESTRO-BLAS (Section 3.3), which uses MAESTRO's analytical equations and hardware backend [20]. The S2 buffers are double-buffered to allow prefetches of the next tiles from memory while the current tiles are being computed. The reported energy in our evaluations is for the on-chip data accesses and movement, since the total off-chip data movement to fetch all operands and write all outputs remains similar across mappings [26].

5.2 Search Space Pruning & Mapping Candidates Reduction

We begin by evaluating search space pruning in FLASH. Given M=256, N=256 and K=256, a MAERI-style mapping with <m, n, k> loop order, and hardware constraints such as S1 buffer size=0.5 KB, S2 buffer size=100 KB, NoC bandwidth=32 GB/s and total number of PE=256, the total number of possible tile size

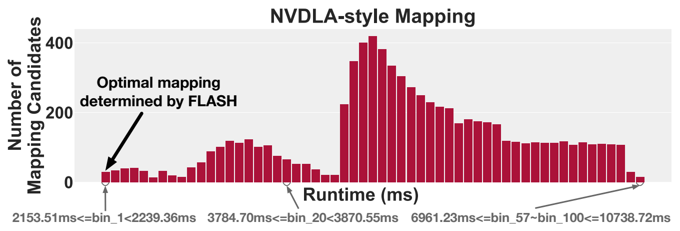


Fig. 7: Histogram of the projected runtime of the pruned mapping candidates for a NVDLA-style STT_TTS-NKM mapping for a GEMM of size (8192×8192)×(8192×8192). In this case, FLASH generates a total of 7,387 mapping candidates. Each bin holds the uniform width (85.85 ms) of runtime and each bar represents the number of mapping candidates included in each bin.

combinations would be 7,250,826,667 sets and would take ∼9.3 hours to even generate the mapping candidates for MAESTRO-BLAS if there are no further constraints. In order to reduce the search space of tiling, we first came up with the tile size constraints for each dimension of the matrices which can be obtained by Equations 3 and 4. By effectively pruning the search space based on the candidate tile size constraints, we could obtain a total of 14,992,384 sets of tile size combinations which requires just 27.75 seconds to generate the candidate inputs. In this instance, FLASH decreases the number of mapping candidates by a factor of 483.63 and reduces the time for generating mapping candidates for MAESTRO-BLAS by 99.9% on a standard laptop. Similar results were observed for other workloads as well. We omit these experiments as evaluating the baseline candidates will require unreasonable amount of time. Note that this demonstrates the effect of pruning only based on the tile size. FLASH also uses the hardware constraints to restrict the search space at the outer loop level which results in even better performance as the number of candidates grow even further.

Fig. 7 demonstrates effectiveness and importance of FLASH with a different experiment. It shows an NVDLA-style GEMM mapping with a <n, k, m> loop order for every cluster size, a total of 7,387 pruned mapping candidates are grouped into 100 bins based on their projected runtime values from MAESTRO-BLAS. FLASH selected mapping is in the bin with the lowest runtime. The figure also demonstrates that a "bad" mapping for this particular accelerator can be up to 4.02× slower than the best mapping. We also ran random sampling [26] and found that FLASH consistently provided the same or better quality of mappings. *To summarize, FLASH reduces the time required for generating mapping candidates through the pruning phase while maintaining low projected runtime*. We plan to explore the multi-objective problem of choosing the mapping that is good in more than one quantity of interest in the future.

5.3 Impact of FLASH Tiling

We demonstrate the impact of tiling chosen by FLASH for MAERI-style mapping for different loop orders. Table 5 compares the number of buffer accesses of the two scratchpads for non-tiled and tiled MAERI-style mappings running workload VI with different loop orders. The number of total S2 accesses with tiled mappings was significantly smaller compared to that of non-tiled mappings for every loop order because the tiled version exploits data reuse using different strategies based on the loop order. ***Overall tiling reduces runtime by 94% and energy reduces by 96% for the <m, n, k> loop order***. Similar improvements are seen with other loop

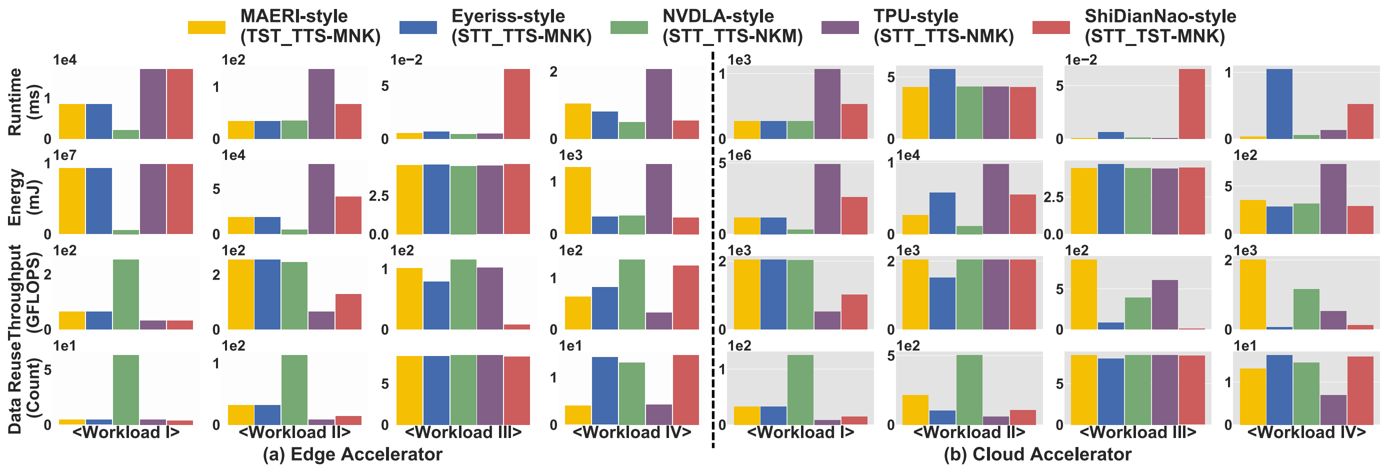


Fig. 8: Runtime, energy, throughput, and data reuse of five mappings listed in Table 2 on (a) edge and (b) cloud accelerators. We apply the best performing cluster size (λ) and tile size (T) for each mapping. The amount of data reuse is calculated by total number of S1 buffer accesses/total number of S2 buffer accesses. One can observe a correlation of data reuse to energy.

TABLE 5: The impact of tiling on the number of buffer accesses, runtime, and energy. We use non-tiled (**NT**) and tiled (**T**) MAERI-style mappings for different loop orders on workload VI and the edge configuration.

Loop Orders	NT/T	Total S1 Access			Total S2 Access			Performance	
		Matrix A	Matrix B	Matrix C	Matrix A	Matrix B	Matrix C	Run time (ms)	Energy (mJ)
$<m,n,k>$	**NT**	3.3E7	6.6E7	6.7E7	2.6E5	3.3E7	2.6E5	2.23	570.02
	T	3.3E7	3.3E7	6.7E7	2.6E5	4.7E5	6.5E5	**0.13**	**21.22**
$<n,m,k>$	**NT**	6.7E7	3.3E7	6.7E7	3.3E7	1.3E5	2.6E5	2.23	570.02
	T	3.4E7	3.3E7	6.7E7	1.1E6	1.3E5	1.1E6	**0.13**	**37.76**
$<m,k,n>$	**NT**	3.3E7	6.7E7	6.7E7	2.6E5	3.3E7	3.3E7	1.31	1132.31
	T	3.3E7	3.3E7	6.7E7	2.6E5	4.7E5	2.6E5	**0.13**	**14.61**
$<n,k,m>$	**NT**	6.7E7	3.3E7	6.7E7	3.3E7	1.3E5	3.3E7	1.31	1132.31
	T	3.4E7	3.3E7	6.7E7	6.0E5	1.3E5	2.6E5	**0.13**	**14.61**
$<k,m,n>$	**NT**	3.3E7	3.3E7	6.7E7	2.6E5	1.3E5	3.3E7	1.31	571.12
	T	3.3E7	3.3E7	6.7E7	2.6E5	1.3E5	1.1E6	**0.13**	**24.26**
$<k,n,m>$	**NT**	5.0E7	3.3E7	6.7E7	1.6E7	1.3E5	3.3E7	1.31	852.26
	T	3.3E7	3.3E7	6.7E7	2.6E5	1.3E5	6.5E5	**0.13**	**15.44**

orders. Since S2 scratchpad accesses consume significantly more energy compared to S1 access or computation, the reduction in S2 accesses leads to dramatic improvements in energy costs. The runtime is also improved by reducing the amount of traffic between a PE and the S2 scratchpad, reducing the on-chip communication delay of an accelerator. The impact of tiling is more significant than that of loop order as 91.25% runtime reduction is observed by tiling on average while loop orders with the best and the worst runtime show 0.8% difference in runtime within tiled mappings. Similar results were observed for tiling for other accelerators as well. Tiled versions with the tile size chosen by FLASH were used for all mappings in the rest of the paper.

5.4 Evaluation of Accelerators and Workloads

The reduced search space based on the tile sizes and hardware constraints with FLASH in addition to having a fast analytical model in MAESTRO-BLAS allows us to study a larger space with five accelerators, two configurations, and six workloads for the pruned loop order, cluster size and tile sizes. We summarize these results in this sub-section.

The Impact of Matrix Shape. Fig. 8 shows the runtime, energy, throughput, and data reuse of five mappings for four selected workloads to show the impact of matrix sizes and shapes. *These experiments use a fixed loop order for fair comparison.*

NVDLA-style mapping does better than other mappings, 81.0% lower runtime and 93.8% less energy on the edge accelerator for square matrices (workload I). However, on the cloud accelerator, all the mappings except ShiDianNao-style mapping achieve near-peak throughput (2 TFLOPS) and similar runtime. NVDLA-style mapping results in 87.8% less energy, on average across mappings. *When loop orders are fixed, NVDLA-style mapping scheme is the best among five mapping schemes for the square matrices (workload I).*

Such results on square matrices are mainly based on two factors; loop order and tile size. The loop order with K at the inner-most position requires data tiles on both matrices A and B because both of them are coupled with dimension K. That is, it is hard to leverage any *data reuse* if we place dimension K at the inner-most position, which leads to bad energy efficiency. The second aspect is the tile size that determines the balance between computation and communication delay in combination with loop order and NoC bandwidth [27]. If the communication delay for data tile fetch is longer than the computation delay of computation tiles on sub-clusters (or, PEs), the communication latency hiding fails, which leads to significant runtime increase. We observe such cases on edge accelerators running MAERI-, Eyeriss-, and TPU-style mappings; they show near-peak bandwidth when the NoC bandwidth is larger in the cloud accelerator.

Short-and-fat matrices in workloads II and III (i.e., $K >> M$ and N) show a different trend based on the skewness of the aspect ratio of matrices. For the workload II, with aspect ratio of 1:8 between M/N and K, MAERI-style and Eyeriss-style mappings provide the lowest runtime, which is 57.1% less compared to other mappings, on average. However, based on the impact of loop order, the mappings requires 73.6% more energy compared to NVDLA-style which is the most energy efficient mapping across edge and cloud accelerators. The energy consumption of Eyeriss-style mapping on cloud accelerator is larger than that on edge accelerator because the tiling parameters are changed and it affects the degree of spatial reuse via multi-casting.

Workload IV is GEMM of a short-and-fat matrix A and a tall-and-skinny matrix B. We observe NVDLA-style and MAERI-style mappings provide the lowest runtime on edge and cloud accelerators, respectively. *This use case demonstrates the success of workload and architecture-aware tiling strategy*. The large skew

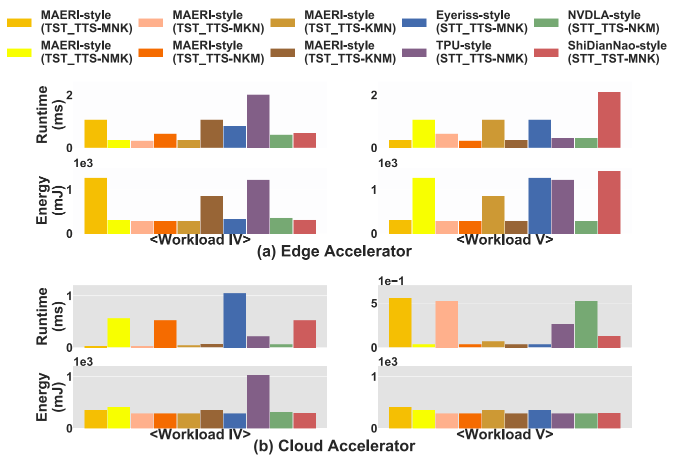


Fig. 9: Performance comparison of the mapping variants by varying all feasible loop orders for MAERI-style (TST_TTS) mapping on two workloads (IV and V) and two accelerators (edge and cloud).

in dimension (M:N=1:1024) results in an extreme tiling strategy for MAERI-style mapping that maximizes data reuse on the smaller matrix A, which significantly reduces the number of expensive S2 buffer accesses.

Next, we identify that the amount of data reuse directly impacts energy usage with a negative correlation. This is because the number of S2 accesses dominate the on-chip energy [20], and data reuse indicates the reduction of the number of S2 accesses. We also observe that high data reuse counts led to high throughput across workloads except workload III on ShiDianNao-style mapping. This is because data reuse and throughput do not have a direct correlation. Also, an output stationary accelerator is not an ideal choice when the size of output matrix C is small as workload III.

We observe that data reuse dramatically differs between mappings depending on the matrix sizes and shapes. For example, NVDLA-style mapping was better suitable for workload I than III (7.5× more data reuse). However, the trend is not the same for all the mapping styles. For MAERI-style mapping, we observe the opposite trend from NVDLA-style mapping; 1.9× more data reuse on workload III than workload I. This is partially due to microarchitecture differences, but fixed loop orders also play a role. Such results show the relationship between mapping and workload is not trivial, which requires consideration of workload, mapping, and architecture. Our FLASH will allow architects to systemically evaluate all the options.

NVDLA-style performs the best across the workloads and accelerators, providing 69.8% lower runtime and 92.3% less energy, on average. However, the non-square workloads prefer different mappings, which implies that no single mapping is ideal for all the workloads. FLASH enables adapting the mappings for such workloads and selects the best performing mapping for each workload. This can provide 4.4% additional runtime and 0.7% energy improvements compared to the mapping (NVDLA-style) optimized for the average case. In the cloud configuration, *MAERI-style mapping achieves the peak FLOPS in three out of four workloads shown here.* The results could change significantly if we fully exploit the flexibility in loop order.

The Impact of Loop Order. Fig. 9 presents the runtime and energy cost of all the evaluated mappings varying six different loop orders for MAERI-style mapping across workloads IV and V. *We observe the loop order has significant impact on both runtime and energy in different degrees depending on the mapping scheme*. For example, for the workload IV, energy usage reduces by 76.5% on MAERI-style mapping (edge configuration) when switched from <m, n, k> to <n, m, k> loop order. Furthermore, on the cloud configuration, MAERI-style mapping with <m, k, n> and <n, k, m> loop orders on workload IV results in 0.03 ms and 0.52 ms of runtime, respectively. The trend reverses in workload V because workloads IV and V are transposes.

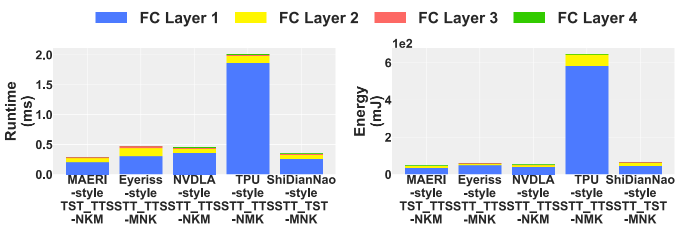


Fig. 10: Performance comparison of the mapping variants on four GEMM workloads (i.e., fully-connected layers) involving Deep Neural Networks and edge accelerator. A fully-connected layer performs GEMM of size (batch size×# of nodes in current layer)×(# of nodes in current layer×# of nodes in the next layer).

The preference to loop orders differs by the mapping schemes and workloads. For example, MAERI-style mapping with <m, n, k> loop order on workload IV and cloud accelerator achieves 0.03 ms of runtime while Eyeriss-style with <m, n, k> achieves 1.05 ms of runtime. The preference toward loop order based on workload and mapping scheme implies that the flexibility in loop order can provide significant runtime and energy benefits. Across all the mappings and workloads IV and V on the edge accelerator, we observe that *flexible loop order selected from FLASH provides 49.9% runtime and 49.7% energy reduction* across mapping schemes compared to the average-workload-optimized fixed loop order of each mapping scheme. *On the cloud accelerator, the potential benefits were more significant on runtime (85.6% improvements) than energy benefits (26.2%) on average*. We also swept the cluster size across the accelerators. We do not show results, in the interest of space, but found that it affects utilization, which in turn affects runtime and energy (up to 42% in our results).

Performance on Deep Neural Networks. As GEMM accounts for approximately 90% of the total number of operations while training/testing DNNs such as fully connected Multi-Layer Perceptron (MLP), we also evaluated GEMM operations for inference of MLP model on different accelerators. The MLP model we used consists of an input layer, three hidden layers and an output layer, and 512, 256 and 128 nodes for each hidden layer. Fig. 10 shows the runtime and energy cost across the various mappings for performing GEMM operations involved in inference of DNNs.

In general, batch data processing is used for DNN training/inference. Therefore GEMM kernel is used on batched data between one layer to the next. For example, using a real-world MNIST image dataset, a fully-connected (FC) layer 1 shown in Fig. 10 connects 28×28=784 nodes in input layer to 512 nodes in the first hidden layer. Hence, FC layer 1 corresponds to the first GEMM operation that multiplies an input matrix of size (128×784) and a weight matrix of size (784×512) where the batch size is set to 128. Similarly, FC layer 4 computes GEMM of size (128×128)×(128×10) to connect 128 nodes in the third hidden layer to 10 nodes (classes) in output layer. With these four GEMM workloads we used for four FC layers in MLP model. In these use cases, MAERI-style and ShiDianNao-style mappings provide lower runtime and MAERI-style and NVDLA-style provide higher energy efficiency relative to the other mappings.

Summary. We summarize our key observations based on all the experiments:

- FLASH reduces the number of mapping candidates that needs to be considered by the analytical model by up to 480$\times$ and still finds a mapping that reduces runtime.
- The new tiling strategy for spatial accelerators significantly reduces runtime and energy usage.
- The evaluation of accelerators on several workloads show loop order is critical for runtime and energy in different degrees for each mapping scheme and workload. Matrix shapes also play an important role.
- Flexible mapping accelerators (e.g., MAERI [7]), with a framework like FLASH can provide significant runtime and energy improvements compared to the best mapping optimized for the average case, showing 65.3% runtime and 30.1% energy improvements on average.

6 Related Work

Mappers for Spatial Accelerators: Most prior works focus on developing mappers specific to their architectures. For example, mRNA [28] for MAERI [7], Auto-TVM [29] for the GEMM core of the VTA architecture [30] limiting their applicability to generic spatial accelerators. Mapping optimizers such as Interstellar [31], dMazeRunner [32] are specific to convolutions and fix certain aspects of the dataflow (such as choice of parallel loops and loop order), constraining the search space. To the best of our knowledge, Timeloop [26] is the only framework that considers all aspects of a mapping for GEMM kernels on a flexible spatial accelerator. However, it employs either an exhaustive linear search or a random sampling-based heuristic.

GEMM Accelerators: Pedram et al., [33] proposed a GEMM accelerator called Linear Algebra Core (LAC) which consists of a $\sqrt{P} \times \sqrt{P}$ 2D grid of PEs. LAC accelerator is based on SUMMA algorithm [34]. which can be considered as a tiled GEMM with $<k, m, n>$ loop order with specific multicast between rows and columns of PEs. Pedram et al., [24] further proposed a multi-core like GEMM accelerator called Linear Algebra Processor (LAP) with multiple LACs and tiled matrices A and B. Each LAC acquires a disjoint tile $A_{LAC\_id,k}$ and a shared tile $B_{k,j}$ so as to ensure that all the LACs in LAP are able to perform GEMM simultaneously. However, LAP is limited to the SUMMA algorithm and its mapping style. In contrast, we focus on several accelerators and map several GEMM variations to them.

Auto-tuning/Tiling for BLAS: Auto-tuning approaches such as ATLAS [35], SPIRAL [36] are popular in generating efficient, optimized, and portable codes for BLAS operations over a variety of platforms. For instance, SPIRAL focuses on auto-tuning of compiler-generated codes. ATLAS uses auto-tuning techniques for loop tiling optimizations, and also for exploring recursion-based alternative implementation choices. The broader ideas of ATLAS to choose between block based algorithms for portability and specific parameter tuning are also described as self-adapting linear algebra [37]. Instead of generating different micro-kernels to choose tiling as they do, we use a cost model to evaluate the variants. Recent work has explored tiling and batching of GEMM on GPUs [38]. Libraries like CUTLASS [39] use a hierarchical tiling for a temporal architecture. The tiling approach we use is similar to the block based methods [35] and a variation of Goto et al. [40] as the accelerator hardware is far simpler (e.g., no TLB). An analytical derivation of tile size is possible in this work as we were able to focus on few hardware parameters that is common among all the accelerators. As accelerators diverge we might have adapt some of the techniques from the auto-tuning approaches as well. Most of these past work limit themselves one popular loop order for CPU architectures. We derive tile sizes for all loop orders instead of one popular loop order, as the architectures are evolving.

7 Conclusion

We develop a framework FLASH for evaluating spatial accelerators via the GEMM kernel. FLASH considers all aspects of the mapping (tile sizes and dataflow for temporal, spatial and spatio-temporal reuse) within the microarchitectural constraints from the accelerator (number of PEs, NoC capability, buffer hierarchy), prunes the search space based on these constraints and additional heuristics, to produce optimized mappings for GEMM. The experimental results show the importance of different co-design considerations computer architects and algorithm designers have to take. This comprehensive study lays the first steps for a heterogeneous HPC node with these accelerators that can address the need of machine learning applications and computational science applications. The ideas in this work could lead to future work studying other dense and sparse ML and CSE kernels over accelerators.

References

[1] N. P. Jouppi *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA)*, 2017.

[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 deep learning accelerator (nvdla)," 2018, http://nvdla.org.

[5] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 367–379, 2016.

[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 *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, 2015, pp. 92–104.

[7] H. Kwon, A. Samajdar, and T. Krishna, "MAERI: Enabling flexible dataflow mapping over DNN accelerators via reconfigurable interconnects," *ACM SIGPLAN Notices*, vol. 53, no. 2, pp. 461–475, 2018.

[8] K. Rocki, D. Van Essendelft, I. Sharapov, R. Schreiber, M. Morrison, V. Kibardin, A. Portnoy, J. F. Dietiker, M. Syamlal, and M. James, "Fast stencil-code computation on a wafer-scale processor," *arXiv preprint arXiv:2010.03660*, 2020.

[9] T. Louw and S. McIntosh-Smith, "Using the graphcore ipu for traditional hpc applications," EasyChair, Tech. Rep., 2021.

[10] M. Emani, V. Vishwanath, C. Adams, M. E. Papka, R. Stevens, L. Florescu, S. Jairath, W. Liu, T. Nama, and A. Sujeeth, "Accelerating scientific applications with sambanova reconfigurable dataflow architecture," *Computing in Science & Engineering*, vol. 23, no. 2, pp. 114–119, 2021.

[11] B. Kågström, P. Ling, and C. Van Loan, "GEMM-based level 3 BLAS: High-performance model implementations and performance evaluation benchmark," *ACM Transactions on Mathematical Software (TOMS)*, vol. 24, no. 3, pp. 268–302, 1998.

[12] K. Goto and R. Van De Geijn, "High-performance implementation of the level-3 BLAS," *ACM Transactions on Mathematical Software (TOMS)*, vol. 35, no. 1, pp. 1–14, 2008.

[13] 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.

[14] T. A. Davis, S. Rajamanickam, and W. M. Sid-Lakhdar, "A survey of direct methods for sparse linear systems," *Acta Numerica*, vol. 25, pp. 383–566, 2016.

[15] E. Bavier, M. Hoemmen, S. Rajamanickam, and H. Thornquist, "Amesos2 and belos: Direct and iterative solvers for large sparse linear systems," *Scientific Programming*, vol. 20, 1970.

[16] A. Abdelfattah, A. Haidar, S. Tomov, and J. Dongarra, "Performance, design, and autotuning of batched GEMM for GPUs," in *International Conference on High Performance Computing*. Springer, 2016, pp. 21–38.

[17] M. Howard, T. C. Fisher, M. F. Hoemmen, D. J. Dinzl, J. R. Overfelt, A. M. Bradley, and K. Kim, "Employing multiple levels of parallelism for cfd at large scales on next generation high-performance computing platforms." Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), Tech. Rep., 2018.

[18] K. Kim, T. B. Costa, M. Deveci, A. M. Bradley, S. D. Hammond, M. E. Guney, S. Knepper, S. Story, and S. Rajamanickam, "Designing vector-friendly compact BLAS and LAPACK kernels," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2017, pp. 1–12.

[19] H. Kwon, A. Samajdar, and T. Krishna, "Rethinking nocs for spatial neural network accelerators," in *2017 Eleventh IEEE/ACM International Symposium on Networks-on-Chip (NOCS)*, 2017.

[20] 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 *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 754–768.

[21] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," in *International Symposium on Computer Architecture (ISCA)*, 2016.

[22] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[23] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE journal of solid-state circuits*, vol. 52, no. 1, pp. 127–138, 2016.

[24] A. Pedram, R. A. Van De Geijn, and A. Gerstlauer, "Codesign tradeoffs for high-performance, low-power linear algebra architectures," *IEEE Transactions on Computers*, vol. 61, no. 12, pp. 1724–1736, 2012.

[25] "MAESTRO: An open-source infrastructure for modeling dataflows within deep learning accelerators," http://maestro.ece.gatech.edu/, 2020.

[26] A. Parashar, P. Raina, Y. S. Shao, Y.-H. 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.

[27] R. Guirado, H. Kwon, E. Alarcón, S. Abadal, and T. Krishna, "Understanding the impact of on-chip communication on DNN accelerator performance," *26th IEEE International Conference on Electronics Circuits and Systems (ICECS)*, 2019.

[28] Z. Zhao, H. Kwon, S. Kuhar, W. Sheng, Z. Mao, and T. Krishna, "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)*, 2019.

[29] 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 12th USENIX Conference on Operating Systems Design and Implementation (OSDI)*, 2018, pp. 579–594.

[30] T. Moreau, T. Chen, L. Vega, J. Roesch, E. Q. Yan, L. Zheng, J. Fromm, Z. Jiang, L. Ceze, C. Guestrin, and A. Krishnamurthy, "A Hardware-Software Blueprint for Flexible Deep Learning Specialization," *IEEE Micro*, vol. 39, no. 5, pp. 8–16, 2019.

[31] 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.

[32] S. Dave, Y. Kim, S. Avancha, K. Lee, and A. Shrivastava, "DMazeRunner: Executing Perfectly Nested Loops on Dataflow Accelerators," *ACM Trans. Embed. Comput. Syst.*, vol. 18, 2019.

[33] A. Pedram, A. Gerstlauer, and R. A. Van De Geijn, "A high-performance, low-power linear algebra core," in *ASAP 2011-22nd IEEE International Conference on Application-specific Systems, Architectures and Processors*. IEEE, 2011, pp. 35–42.

[34] R. A. Van De Geijn and J. Watts, "SUMMA: Scalable universal matrix multiplication algorithm," *Concurrency: Practice and Experience*, vol. 9, no. 4, pp. 255–274, 1997.

[35] R. C. Whaley, A. Petitet, and J. J. Dongarra, "Automated empirical optimizations of software and the atlas project," *Parallel computing*, vol. 27, no. 1-2, pp. 3–35, 2001.

[36] F. Franchetti, Y. Voronenko, P. A. Milder, S. Chellappa, M. R. Telgarsky, Hao Shen, P. D'Alberto, F. de Mesmay, J. C. Hoe, J. M. F. Moura, and M. Püschel, "Domain-specific library generation for parallel software and hardware platforms," in *2008 IEEE International Symposium on Parallel and Distributed Processing*, 2008, pp. 1–5.

[37] J. Demmel, J. Dongarra, V. Eijkhout, E. Fuentes, A. Petitet, R. Vuduc, R. C. Whaley, and K. Yelick, "Self-adapting linear algebra algorithms and software," *Proceedings of the IEEE*, vol. 93, no. 2, pp. 293–312, 2005.

[38] X. Li, Y. Liang, S. Yan, L. Jia, and Y. Li, "A coordinated tiling and batching framework for efficient GEMM on GPUs," in *Proceedings of the 24th Symposium on Principles and Practice of Parallel Programming*, 2019, pp. 229–241.

[39] A. Kerr, D. Merrill, J. Demouth, and J. Tran, "Cutlass: Fast linear algebra in cuda c++," *NVIDIA Developer Blog*, 2017.

[40] K. Goto and R. A. v. d. Geijn, "Anatomy of high-performance matrix multiplication," *ACM Transactions on Mathematical Software (TOMS)*, vol. 34, no. 3, pp. 1–25, 2008.

APPENDIX

We include finer level details in this appendix.

Mapping Candidate Generation Steps: Algorithm 2 shows how mapping candidates are generated in FLASH. FLASH generates mapping candidates given the type of accelerators, hardware parameters, and the sizes of M, N, and K as inputs. Based on the accelerator chosen by the user, FLASH first determines three parameters – dataflow directive order, all feasible loop orders, and all possible cluster sizes that can be explored in the mapping candidates according to the hardware constraints described in Table 2 in Section 3.1. After choosing these three parameters, FLASH computes the candidate tile sizes using problem dimensions and hardware parameters. These tile sizes are used to prune a huge number of possible mappings. Finally, FLASH generates the pruned mapping candidates descriptions as inputs to MAESTRO-BLAS and analyzes the results of MAESTRO-BLAS to choose the best GEMM mapping based on projected runtime.

Tile Size Constraints: FLASH is able to restrict the search space for different accelerators, based on the input dimension and hardware constraints. While one example derivation and the corresponding constraints for tile sizes are shown in Section 4, Table 6 shows all the tile sizes for all the mappings. Restricting the search space using these constraints helps FLASH in choosing the near-optimal mapping.

Algorithm 2: Mapping Candidates Generator in FLASH

```
Input: $M, N, K$: matrix dimensions, $Arch$: type of accelerator, $P$: total number of PEs, $\alpha$: size of S1 buffer, $\beta$: size of S2 buffer, $\lambda$: size of cluster
Output: Mapping_Candidates: mapping descriptions of all mapping candidates
dataflow_candidates ← get_candidate_LoopOrders_ClusterSzs($Arch$, $P$);
num_LoopOrders ← get_num_LoopOrders(dataflow_candidates);
num_ClusterSzs ← get_num_ClusterSzs(dataflow_candidates);
for LoopOrder_id = 1 to num_LoopOrders do
    for ClusterSz_id = 1 to num_ClusterSzs do
        [$\lambda$, DirectiveOrder, outer_LoopOrder, inner_LoopOrder] ← get_dataflow(LoopOrder_id, ClusterSz_id, dataflow_candidates);
        [$T_M^{out}$, $T_N^{out}$, $T_K^{out}$] ← calculate_candidate_outer_Tile_Size($\lambda$, DirectiveOrder, outer_LoopOrder, $\beta$, $P$, $M$, $N$, $K$);    ▷ Eq. 3
        [$T_M^{in}$, $T_N^{in}$, $T_K^{in}$] ← calculate_candidate_inner_Tile_Size($T_M^{out}$, $T_N^{out}$, $T_K^{out}$, $\lambda$, DirectiveOrder, inner_LoopOrder, $\alpha$, $P$, $M$, $N$, $K$);    ▷ Eq. 4
        pruned_TileSzs ← get_pruned_Tile_Sizes($T_M^{out}$, $T_N^{out}$, $T_K^{out}$, $T_M^{in}$, $T_N^{in}$, $T_K^{in}$, DirectiveOrder, $\alpha$, $\beta$);
        Mapping_Candidates ← generate_mapping_candidates(pruned_TileSzs, $\lambda$, DirectiveOrder, outer_LoopOrder, inner_LoopOrder, $M$, $N$, $K$);
```

TABLE 6: The candidate tile size constraints for five different mapping schemes for different accelerators where P, α, β and λ denote total number of PEs, size of S1 buffer, size of S2 buffer and size of cluster, respectively.

Tile Sizes	Eyeriss-style	NVDLA-style	TPU-style	ShiDianNao-style	MAERI-style
$\boldsymbol{T_M^{out}}$	$T_M^{out} = \frac{\lambda M}{P}$	$1 \le T_M^{out} \le \frac{\sqrt{N^2(\lambda+1)^2+2\beta\lambda}-N(\lambda+1)}{2\lambda}$	$1 \le T_M^{out} \le \frac{\sqrt{N^2(\lambda+1)^2+2\beta\lambda}-N(\lambda+1)}{2\lambda}$	$T_M^{out} = \frac{\lambda M}{P}$	$1 \le T_M^{out} \le \sqrt{\frac{\beta}{2}+N^2}-N$
$\boldsymbol{T_N^{out}}$	$1 \le T_N^{out} \le \frac{\sqrt{M^2(\lambda+1)^2+2\beta\lambda}-M(\lambda+1)}{2\lambda}$	$T_N^{out} = \frac{\lambda N}{P}$	$T_N^{out} = \frac{\lambda N}{P}$	$1 \le T_N^{out} \le \frac{\sqrt{M^2(\lambda+1)^2+2\beta\lambda}-M(\lambda+1)}{2\lambda}$	$T_N^{out} = \frac{NT_K^{out}}{P}$
$\boldsymbol{T_K^{out}}$	$1 \le T_K^{out} \le \frac{\sqrt{M^2(\lambda+1)^2+2\beta\lambda}-M(\lambda+1)}{2\lambda}$	$1 \le T_K^{out} \le \frac{\sqrt{N^2(\lambda+1)^2+2\beta\lambda}-N(\lambda+1)}{2\lambda}$	$1 \le T_K^{out} \le \frac{\sqrt{N^2(\lambda+1)^2+2\beta\lambda}-N(\lambda+1)}{2\lambda}$	$1 \le T_K^{out} \le \frac{\sqrt{M^2(\lambda+1)^2+2\beta\lambda}-M(\lambda+1)}{2\lambda}$	$1 \le T_K^{out} \le \sqrt{\frac{\beta}{2}+N^2}-N$
$\boldsymbol{T_M^{in}}$	$1 \le T_M^{in} \le \sqrt{\frac{\alpha}{2}+(T_K^{out})^2}-T_K^{out}$	$1 \le T_M^{in} \le \sqrt{\frac{\alpha}{2}+(T_K^{out})^2}-T_K^{out}$	$1 \le T_M^{in} \le \sqrt{\frac{\alpha}{2}+(T_K^{out})^2}-T_K^{out}$	$1 \le T_M^{in} \le \sqrt{\frac{\alpha}{2}+(T_N^{out})^2}-T_N^{out}$	$1 \le T_M^{in} \le \sqrt{\frac{\alpha+2}{2}}-1$
$\boldsymbol{T_N^{in}}$	$1 \le T_N^{in} \le \sqrt{\frac{\alpha}{2}+(T_K^{out})^2}-T_K^{out}$	$1 \le T_N^{in} \le \sqrt{\frac{\alpha}{2}+(T_K^{out})^2}-T_K^{out}$	$1 \le T_N^{in} \le \sqrt{\frac{\alpha}{2}+(T_K^{out})^2}-T_K^{out}$	$T_N^{in} = T_N^{out}$	$1 \le T_N^{in} \le \sqrt{\frac{\alpha+2}{2}}-1$
$\boldsymbol{T_K^{in}}$	$T_K^{in} = T_K^{out}$	$T_K^{in} = T_K^{out}$	$T_K^{in} = T_K^{out}$	$1 \le T_K^{in} \le \sqrt{\frac{\alpha}{2}+(T_N^{out})^2}-T_N^{out}$	$T_K^{in} = 1$