Principle-based Dataflow Optimization for Communication Lower Bound in Operator-Fused Tensor Accelerator

Lei Xu, Chen Yin, Zelong Yuan, Weiguang Sheng, Jianfei Jiang, Qin Wang, Naifeng Jing*
Shanghai Jiao Tong University, Shanghai, China
xlxl1027@sjtu.edu.cn, sjtuj@sjtu.edu.cn

***Abstract*—Although design space exploration (DSE) is good at finding dataflow for optimal memory access in tensor accelerators, it is very timing-consuming and lacks architecture insight. In this study, we for the first time propose several principles for dataflow optimization that provides lower bound of memory communication for tensor operators such as matrix multiplication. Through these principles we can calculate the best tiling, scheduling and mapping for both intra- and inter-operator dataflow. In addition, we can identify all the tensor-wise opertor fusion that are profitable in memory communication, so we propose FuseCU, a new architecture that supports these profitable fusion which can be applied to existing spatial architectures for data movement saving. Experimental results show that FuseCU delivers 63.6%, 62.4% and 38.7% data movement saving and 1.33×, 1.25× and 1.14× speedup compared to the TPUv4i, Gemmini and Planaria designs without increasing buffer size or bandwidth. Additionally, FuseCU is open-sourced.**

***Index Terms*—dataflow, tensor, principle, fusion, processing elements.**

I. Introduction

Memory access is becoming a major bottleneck in performance and a key factor in the energy consumption of tensor applications like Transformers. To address this challenge, various dataflow optimization techniques have been proposed. Prior research has largely focused on intra-operator dataflow, where data movement is optimized within a single tensor operator, especially in convolution and matrix multiplication (MM) operators [1]–[9]. More recently, researchers are shifting towards inter-operator dataflow—also known as operator fusion—to exploit data reuse across operator boundaries [10]–[15]. Unlike intra-operator dataflow, which repeatedly accesses memory across operator boundaries as illustrated in Fig. 1(a), inter-operator dataflow can enhance data reuse, and potentially eliminate memory communication for intermediate data in an ideal case as in Fig. 1(b).

However, achieving optimal dataflow is nontrivial. Dataflow should taking tiling, scheduling and hardware mapping into overall consideration, and thus create a vast design space. Classic dataflow optimization largely relies on searching-based design space exploration (DSE). However, there are two problems with searching-based optimization:

1) It is time-consuming. If fusion is introduced, the design space expands from a single operator to multiple operators, possibly making search time even unacceptable [14], [15].
2) It lacks architecture insight. As a black box optimization, [11], [13]–[15], searching-based optimization sheds limited insight on architecture innovations.

In this paper, we propose a principle-based dataflow optimization that provides lower bounds for memory communication. It gives optimal dataflow analytically in one-shot. More importantly, these principles offer us new architecture and dataflow insights, so we can propose an efficient operator-fused tensor accelerator called FuseCU, which supports all profitable fusion on tensor operators. To the best of our knowledge, this is the first work to derive the optimal inter-operator dataflow through a principle-based analytical method and the first to unlock the full potential of tensor operator fusion on compute units through architecture innovation. Our key contributions are summarized as follows:

- We propose principle-based dataflow optimization achieving the lower bound for memory communication. These principles are derived from analyzing various dataflow strategies under different tensor and buffer sizes, and directly tell us an efficient dataflow on tensor operators like MM.
- We apply these principles on tensor operator fusion, which guides us to conclude several but all the profitable tensor-wise operator fusion regarding memory communication. This provides us with insights into enabling efficient fusion on compute units through effective mapping and architecture innovations.
- We propose FuseCU, a novel architecture which has sufficient flexibility to support all the profitable fused dataflow. Operators like fused MM can be mapped on a number of PEs without adding any buffers or registers to existing spatial architectures.

We validate our principle-based optimization against the state-of-the-art searching-based dataflow optimizer DAT [15], build FuseCU on TPUv4i [5] and evaluate them using seven attention-based models, which consist of different-sized MM operators. The experimental results show that FuseCU saves 63.6%, 62.4%, and 38.7% data movement compared to TPUv4i [5] and SOTA designs [16], [17], and 1.33×, 1.25×, and 1.14× speedup on average, respectively.

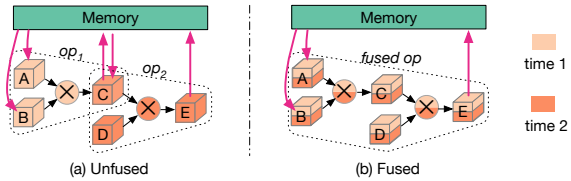

Fig. 1. Operator fusion further reduces memory traffic than unfused operators.

II. Previous Works and Background

A. Dataflow Definition

Dataflow mainly consists of three key parts: tiling, scheduling, and mapping.

- **Tiling** determines the size of the tiles of tensors held in the buffer. As shown in Fig. 2(a), tiling stores only a portion of the whole tensor in the buffer. This tile can aviod memory access by keep data locally. By carefully choosing the tile size of each tensor, better tiling can increase data reuse in buffers.
- **Scheduling** determines the order of tile computations to further exploit data reuse. As shown in Fig. 2(b), different loop orders lead to different tile movements or computation sequences. Memory access can be minimized by keeping unchanged tiles in the buffer. In existing works, this is referred to as the "*stationary*", where unchanged tensor tiles between adjacent computations remain in the buffer.
- **Mapping** determines the assignment of the tiles to the underlying hardware resources. As shown in Fig. 2(c), this involves

*This work is supported by the National Natural Science Foundation of China under Grant No.U24B20165 and No.92264108. Naifeng Jing is the corresponding author.

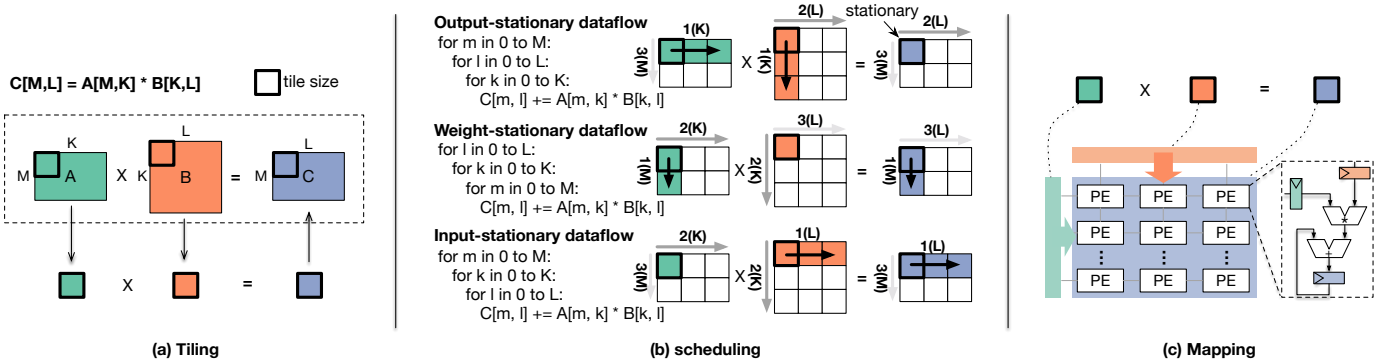

Fig. 2. Three key parts in dataflow for operator $A \times B = C$. We use term X(D) to denote the *scheduling* by loop order "X" on dimension "D" as in the left code. For example in output-stationary (OS), "1(K)" implies the innermost loop on dimension K. We use the arrows to indicate tile movement.

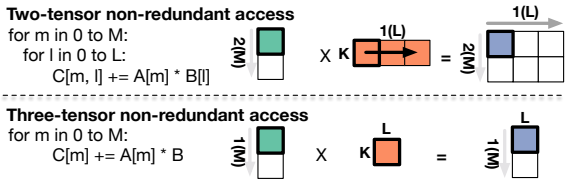

Fig. 3. Two-NRA and Three-NRA dataflow for larger buffer size.

assigning tiles to processing element (PE) array ports and registers, with specific options tailored to the hardware. In this example, C is mapped to registers, while A and B are mapped to the ports, requiring dedicated wiring in the PE.

In practice, the tiling and scheduling optimize the communication between memory and buffer, while mapping optimizes the communication between buffer and PE.

B. Dataflow Optimization

Table I provides a summary of SOTA dataflow optimization frameworks. For intra-operator dataflow, they typically leverage searching-based DSE to optimize tiling, scheduling, and mapping jointly, as each of these aspects impacts the others [1]–[4], [7]. For inter-operator dataflow, several works have proposed tensor operator fusion within memory and employed DSE to select optimal dataflow [11], [12], [18]. Due to the extensive design space, they often separate the mapping phase from tiling and scheduling, or work with incomplete spaces. Additionally, existing works focus on tensor operator fusion on memory without addressing fusion across compute units or PEs, as mapping onto the PE array is hardware-dependent. In summary, existing works generally rely on searching for optimization and overlook compute-level fusion.

III. Principles for Dataflow Optimization

In this section, we propose principles for dataflow optimization, which guild us to select the best dataflow regarding tiling and scheduling for the least memory access (MA), given the specific tensor size of the operator and the buffer size (BS) of the hardware. The principles also guide the best mapping methods in Sec. IV-A. We will use operator MM as an example.

A. Intra-operator Dataflow

For MM, there are three distinct BS that influence optimal dataflow. Ideally, if BS were infinite, all tensors would be fetched only once, achieving the ideal minimal MA for this operator. However, due to limited BS, additional MA is required on the same data, which is termed as redundant access in this paper and should be minimized. As BS increases, the dataflow transitions from single-tensor non-redundant access to two-tensor non-redundant access, and finally to three-tensor non-redundant access.

*1) Single-tensor Non-redundant Access (**Single-NRA**):* For small BS, only the stationary tensor can avoid redundant access. As shown in Fig. 2(b), the stationary tensor is the only one accessed a single time. Two primary decisions for optimal dataflow in this case are selecting the stationary tensor and determining the tiling strategy.

Taking the output-stationary dataflow in Fig. 2(b) as an example, tensor C is kept in the buffer to minimize redundant memory access during adjacent computations, while tensors A and B are re-accessed. Memory accesses for each tensor are calculated as the product of tile sizes and iteration counts. For each dimension D, let T_D represent the tile size. For tensors A and B, the memory accesses are $MK\frac{L}{T_L}$ and $KL\frac{M}{T_M}$ respectively, and for C, it is ML. Thus, total MA can be expressed as:

$$\text{MA} = MKL \times \left(\frac{1}{T_L} + \frac{1}{T_M}\right) + ML \tag{1}$$

To minimize MA, T_L and T_M, which are stationary tensor dimensions, should be maximized. This requires minimal T_K as the total buffer size is limited. Given the BS constraint:

$$T_M T_K + T_K T_L + T_M T_L \leq \text{BS} \tag{2}$$

Minimum MA is achieved at $T_M = T_L = T$. This analysis applies similarly across other stationary dataflow, yielding the same conclusion. Comparing MAs for different stationary dataflow, as in Eq. 1, the initial term is constant as $MKL \times (2/T)$, so the difference lies in the stationary tensor's MA component. With small BS, MA for the stationary tensor is relatively minor compared to others. But, choosing the smallest tensor as stationary minimizes MA.

Principle 1 *Tiling: maximize tile size for stationary tensor dimensions, minimize for non-stationary ones.*
Scheduling: choose the smallest tensor to be stationary.

*2) Two-tensor Non-redundant Access (**Two-NRA**):* With a larger BS, a second tensor can achieve non-redundant access by not tiling one dimension. As shown in Fig. 3, compared to the output-stationary dataflow in Fig. 2(b), the loop over dimension K is removed by keeping this dimension untiled. Consequently, tensors A (input) and C (output) are both accessed without redundancy.

The MA in this dataflow is:

$$\text{MA} = MKL \times \frac{1}{T_M} + MK + ML \tag{3}$$

TABLE I
SUMMARY OF THE SOTA DATAFLOW OPTIMIZERS.

Features	Intra-operator	Inter-operator				
	[1], [3], [6], [7]	Chimera [12]	SET [13]	Flat [11]	DAT [14], [15]	This Work
Full tiling & scheduling optimization space	×	×	×	×	✓	✓
Tiling & scheduling optimization scheme	Searching-based	Searching-based	Searching-based	Searching-based	Searching-based	Principle-based
Mapping optimization scheme	Searching with fixed patterns	Replaceable micro kernels	Not discussed	Not discussed	Not discussed	Principle-based
Fusion medium	No fusion	Memory	Memory	Memory	Memory	Compute unit

To minimize MA, T_M, which is the dimension not in the redundant access tensor, should be maximized under the buffer size constraint, requiring minimal T_L. The new BS constraint becomes:

$$T_M K + K T_L + T_M T_L \leq \text{BS} \tag{4}$$

The untiled dimension size significantly limits T_M, so choosing the smallest dimension to remain untiled is essential.

Principle 2 *Tiling: maximize the tile size for the dimension not in the redundant access tensor, minimize for others. Scheduling: untile/unroll the smallest dimension.*

*3) Three-tensor Non-redundant Access (**Three-NRA**):* With a sufficiently large buffer, all three tensors achieve non-redundant access by leaving two dimensions untiled. As shown in Fig. 3, only one dimension is tiled, and its tile size does not impact MA. To achieve non-redundant access for all three tensors with minimal buffer size, we should untile the smallest tensor.

Principle 3 *Tiling: do not care. Scheduling: untile/unroll the smallest tensor.*

4) Choosing the Best Dataflow: With the tiling and scheduling principles established for each dataflow, the next step is to determine the optimal dataflow based on the BS compared to the tensor size.

To compare Single-NRA and Two-NRA, we derive the relationship between BS and MA under optimal tiling and scheduling for each dataflow according to our priciples and compare the MAs for the same BS. We omit the analysis details due to limited space. This analysis reveals that when the BS is small, Single-NRA outperforms Two-NRA. However, as BS increases, the optimal dataflow shifts from Single-NRA to Two-NRA. The shift point lies within the range $D^2_{\text{min}}/4$ to $D^2_{\text{min}}/2$, where D_{min} denotes the smallest tensor dimension.

The comparison between Two-NRA and Three-NRA is more straightforward: Three-NRA is preferred when the BS is sufficiently large to accommodate the minimal tensor. At this point, each tensor is accessed only once, achieving the ideal minimal MA for intra-operator dataflow.

Therefore, we classify BS into four categories based on their relation to tensor dimension sizes:

$$\begin{aligned}
&\text{Tiny buffer:} && \text{BS} \leq D^2_{\text{min}}/4 && \Rightarrow \text{Single-NRA}\\
&\text{Small buffer:} && D^2_{\text{min}}/4 < \text{BS} \leq D^2_{\text{min}}/2 && \Rightarrow \text{Single/Two-NRA}\\
&\text{Medium buffer:} && D^2_{\text{min}}/2 < \text{BS} \leq \text{Tensor}_{\text{min}} && \Rightarrow \text{Two-NRA}\\
&\text{Large buffer:} && \text{BS} > \text{Tensor}_{\text{min}} && \Rightarrow \text{Three-NRA}
\end{aligned}$$

Note that for small buffers, both Single-NRA and Two-NRA dataflow can be used, with the optimal one selected based on specific conditions. In all other cases, the best dataflow can be directly chosen based on buffer size and tensor dimension sizes.

Example: Consider the MM of $A_{1024,768} \times B_{768,768} = C_{1024,768}$ in BERT [19] by assuming a BS of 512 KB. Given that BS $> D^2_{\text{min}}/2 = 768^2/2 = 294,912$ and BS $< Tensor_{\text{min}} = B = 589,824$, the optimal dataflow is Two-NRA. According to Principle 2, we do not tile the K-dimension, using the same scheduling as shown in the top part of Fig. 3. Next, according to the BS constraint as in Eq. 4, we maximize T_M to 512 and minimize T_L to 1. This analytical dataflow result enables non-redundant access for both tensors A and C, while minimizing memory access for tensor B to $2KL$, which matches the best dataflow searched using DSE as reported in [2], [15].

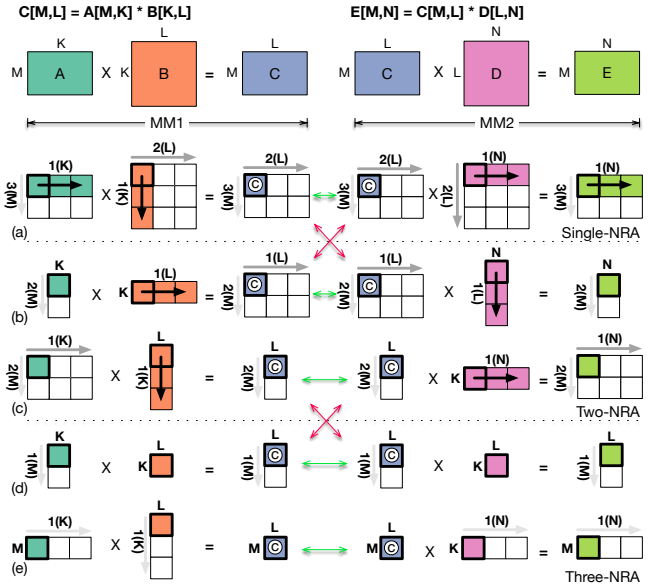

Fig. 4. Dataflow fusion patterns. The green arrows indicate profitable fusion, while the red arrows indicate the fusable but non-profitable fuison.

B. Inter-operator Dataflow

When operator fusion is introduced, the tiling and scheduling of the intermediate tensor need to be considered under the inter-operator scope. For example, in Fig. 4, the tile size and tile movement of tensor C should be identical in MM1 and MM2 if they are fused. This constraint may hinder intra-operator dataflow optimization, leading to significant MA deterioration. Therefore, it is necessary to determine whether the fusion is profitable with respect to MA.

We first give all the dataflow where operator fusion can be applied, than indicate the profitable ones.

1) Fusiablity: Tensor operators can only be fused if there is *no redundant access to the intermediate tensor* within their intra-operator dataflow. Redundant access is prohibited because the intermediate tensor is not stored in memory in fused dataflow.

There are three methods to achieve no redundant access in intra-operator dataflow: ① making the tensor stationary, ② un-tiling a dimension, ③ keeping the entire tensor in buffer. In Single-NRA, only ① is applicable. In Two-NRA, both ① and ② are applicable. In Three-NRA, ② and ③ are applicable.

By permuting and combining these methods, we can derive all feasible dataflow for fusion, as illustrated in Fig.4, which include all fused dataflow in recent works [11], [13]–[15].

As shown in Fig. 4(a), in the Single-NRA dataflow, fusion is possible only by setting MM1 as Output-Stationary (OS) and MM2 as Input-Stationary (IS), ensuring the intermediate tensor C is stationary in both operators. In Fig. 4(b) and (c), for the Two-NRA dataflow, fusion can similarly be achieved using OS-IS, or by unrolling dimension L of tensor C. Finally, as depicted in Fig. 4(d) and (e), in the Three-NRA dataflow, fusion can be enabled by either unrolling a dimension of C or retaining the entire intermediate tensor C on-chip.

There are also fusions for two operators with different NRA dataflow. We indicate them as red arrows in Fig. 4, which use making C stationary to fuse Single-NRA and Two-NRA, and un-tiling L to fuse Two-NRA and Three-NRA.

2) Profitablity: We find that fusion with the same NRA is profitable, while fusion with different NRA is non-profitable.

Although it is possible to fuse MMs with different NRA dataflow, this approach is not profitable. Consider the example of fusing Single-NRA MM1 in Fig. 4(a) with Two-NRA MM2 in (b). According to Principle 1, Single-NRA aims to maximize the dimension L, while according to Principle 2, Two-NRA minimizes L. To enable fusion, we have to choose a medium T_L, which is suboptimal for both MM1 and MM2. Although fusion eliminates the MA of tensor C, it increases the MA for other tensors. As the redundant memory accesses of other tensors dominate the overall MA, this fusion results in increased MA.

In contrast, operators sharing the same NRA dataflow follow consistent tiling principles, allowing fusion without disrupting the optimal intra-operator dataflow. We define this type of fusion as "profitable", as illustrated by the green arrows in Fig. 4.

Principle 4 *Only fuse tensor operators with the same NRA dataflow.*

Principle 1-4 can be extended to other tensor operators, as all tensor operators can be represented as for-loops, verying only on the number of loop levels while sharing consistent derivation. For the fusion of more than two operators, we can apply Principle 4 to each pair of connected operators to determine whether to fuse them.

IV. Tensor Operator Fusion Architecture

While the principles provide the minimal MA and the corresponding dataflow, these dataflow cannot be implemented on processing units (PEs) in conventional architectures.

First, unlike the mapping of intra-operator dataflow, which can be obtained from existing work, there is no work that indicates how to map all the profitable fused dataflow onto PEs.

Second, unlike many tensor accelerators that only support intra-operator dataflow and rely on a single type of stationary and tile size, accelerators designed for tensor operator fusion require *flexible stationary* and *adaptive tile size*.

A. Fused Dataflow Mapping

Mapping needs to decide which dimension maps across time and which dimensions map across PEs [20]. We refer to the tile whose dimensions all map across PEs as the "stationary tile" and the tile with one dimension mapping across time as the "moving tile", as shown in Fig. 5. The stationary tile size have to match the array size. otherwise, it will cause low utilization. While the moving tile size is less constrained.

According to the principles for tiling optimization, we identify two distinct tile sizes for the intermediate tensor C in the profitable fused dataflow in Fig. 4:

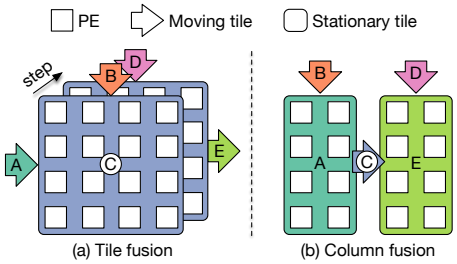

Fig. 5. Fused dataflow mappings.

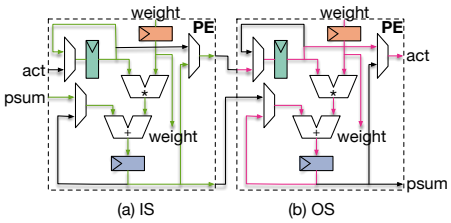

Fig. 6. PE design. Green wire: datapath for IS. Red wire: datapath for OS.

- **Tile-like**, as in Fig. 4(a), (c), and (e), the optimal tile size (ⓒ) is optimized through either maximizing or leaving untiled, according to Principle 1-3 respectively.
- **Column-like**, as in Fig. 4(b) and (d), the optimal tile size has one dimension maximized and the other minimized, according to Principle 2.

The tile-like size is well-suited as stationary tile for mapping across PEs, while the column-like size would cause low utilization if mapped as stationary tile. Therefore, we propose two fused dataflow mappings: *tile fusion* and *column fusion*.

In *tile fusion* (Fig. 5(a)), the tensor tile ⓒ is mapped as stationary tile on PEs. The computation begins with OS for $A \times B = C$, followed by IS for $C \times D = E$.

In *column fusion* (Fig. 5(b)), the PEs are divided into two parts, with the intermediate tensor tile ⓒ placed between them, mapped as moving tile. The first part uses IS, while the second part uses OS. Each column of C (ⓒ), produced by the first part, is consumed by the second part, enabling efficient pipelined computation.

For example, if we need to map the MMs with the corresponding tile sizes: $A(128,1) \times B(1,128) = C(128,128)$ and $C(128,128) \times D(128,1) = E(128,1)$, which is common in Single-NRA fused dataflow, we would use the tile fusion mapping as in Fig. 5(a). If the MMs with tile sizes $A(128,128) \times B(128,1) = C(128,1)$ and $C(128,1) \times D(1,128) = E(128,128)$, which is common in Two-NRA fused dataflow, we would use the column fusion mapping as in Fig. 5(b).

By leveraging both tile and column fusion, we provide comprehensive support for all profitable fused dataflow on PEs.

B. Operator-Fused Tensor Accelerator

To achieve flexible stationary and adaptive tile size, we propose the X-Stationary (XS) PE and FuseCU architecture.

First, we propose a novel XS PE to support flexible stationary. As shown in Fig. 6, additional multiplexers (MUXs) are used to adjust the datapath, supporting both IS and OS dataflow. Weight-Stationary (WS) can also be supported by simply swapping the positions of activations and weights. Column fusion is facilitated by the MUX in the activation output, allowing for selection between the input activation and the quantized result.

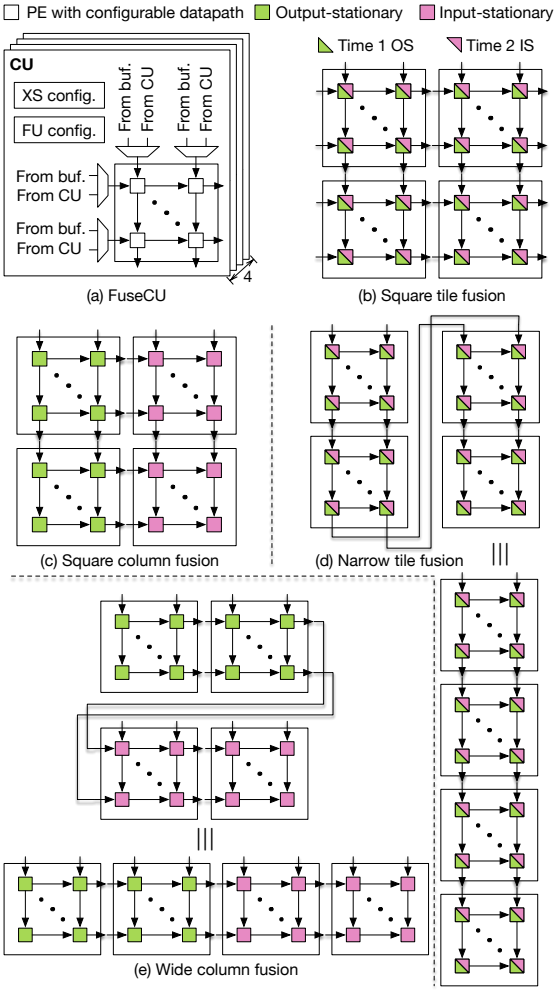

Fig. 7. FuseCU design. Wide tile fuison and narrow column fusion is omited for simplicity.

Next, we address the need for adaptive tile size. Adaptive tile size is essential for the untiled dimension in fused dataflow, as it is determined by the tensor operator.

Assuming we have an $N \times N$ PE array, our principles reveal that the widest untiled dimension size to support is $2N$.

As shown in Fig. 4, the untiled dimension occurs only in Two-NRA and Three-NRA dataflow. From Sec. III-A4, we determine the BS required for these dataflow to be optimal: $BS > D_{\text{min}}^2/4$. Since BS corresponds to the register size now, which is the number of PEs ($N \times N$), we derive $D_{\text{min}} < 2N$ from the inequality $N^2 > D_{\text{min}}^2/4$. This implies that an un-tiling strategy is optimal only when $D_{\text{min}} < 2N$, and therefore, we only need to support untiled dimension sizes smaller than $2N$. This insight guides the architecture design of FuseCU.

We propose FuseCU, which supports adaptable array configurations capable of handling all profitable fused dataflow. As shown in Fig. 7(a), FuseCU consists of four Compute Units (CUs). MUXes are added to the ports of the PE array in each CU, allowing the edge PEs to select data either from memory or adjacent CUs. The XS configuration controls the stationary within PEs, while the Fusion (FU) configuration manages connections between CUs.

Consequently, as shown in Fig. 7(b) and (c), FuseCU can achieve both tile fusion and column fusion by setting corresponding stationary

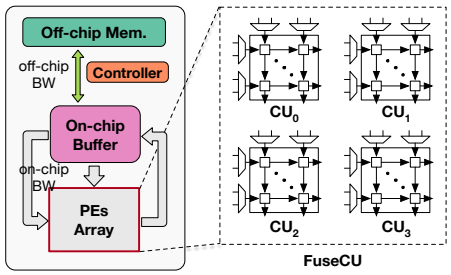

Fig. 8. Spatial accelerator architecture modeling.

TABLE II
TRANSFORMER MODEL PARAMETERS

Model	# of Heads	Seq. Length	Hidden Size
Bert	12	1024	768
GPT-2	12	2048	768
Blenderbot	16	256	1024
XLM	16	1024	2048
DeBERTa-v2	24	1024	1536
LLaMA2	32	4096(256-16K)	4096
ALBERT	64	1024	4096

patterns, and enable adaptable array sizes—square, narrow, and wide—by configuring different connections, as illustrated in Fig. 7(c-e). For instance, in Fig. 7(d), we configure the all CUs as OS-IS, connecting the bottom-left CU with the top-right CU to achieve narrow tile fusion. In Fig. 7(e), we configure the top two CUs as IS and the bottom two as OS, connecting the top-right CU with the bottom-left CU to achieve wide column fusion.

Note that by introducing the configurable array size, we support untiled dimension size of up to $2N$, and with flexible stationary, we enable all optimal inter-operator dataflow on compute units.

FuseCU can be easily applied to existing spatial architectures, as it does not modify any existing logic within the PE array. It simply adds MUX and wires to enable greater flexibility.

V. Evaluation and Results

A. Experiment Setup

Spatial architecture parameters. We evaluate our principles and FuseCU on spatial architectures, as illustrated in Fig. 8. For the memory component, we use various buffer sizes, ranging from 32KB to 32MB, to validate our principles. For the compute component, we use the configuration from TPUv4i [5], setting the total number of PEs to 128 × 128 × 4 and the on-chip bandwidth to 1TB/sec.

Architecture modeling. We adopt DAT [15] for inter-operator dataflow and MAESTRO [2] for intra-operator dataflow to obtain memory access and cycle counts. These are open-source models widely used in dataflow optimization research [3], [7], [21]. For compute modeling, we implement FuseCU in Chisel to generate Verilog RTL, which is open-sourced at https://github.com/lxu28973/fuseCU.git. The RTL is then synthesized using Synopsys Design Compiler to obtain area estimates.

TABLE III
SPATIAL ARCHITECTURE ATTRIBUTES

Platform	Stationary Flex.	Tiling Flex.	Tensor Fusion
TPUv4i	×	low	×
Gemmini	✓	low	×
Planaria	×	high	×
UnfCU	✓	middle	×
FuseCU	✓	middle	✓

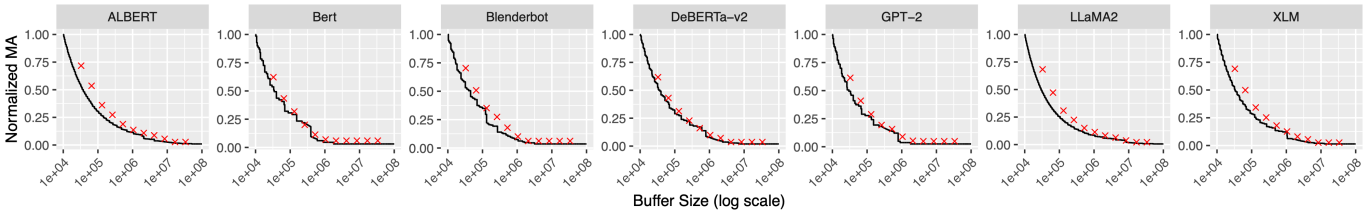

Fig. 9. Normalized memory access validation against DAT. The line indicates the dataflow our optimized, the points indicate the dataflow DAT optimized.

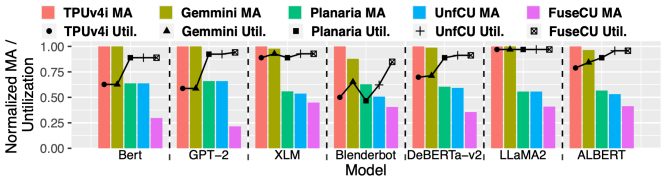

Fig. 10. Comparison of normalized memory access and utilization.

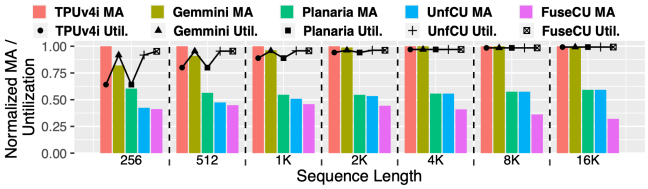

Fig. 11. Comparison of LLaMA2 under difference sequence length.

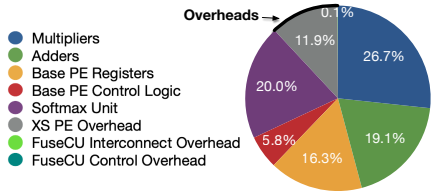

Fig. 12. FuseCU area breakdown and its overheads.

Workloads. We study seven recent open-source attention-based models. The model sizes and input sequence lengths are shown in Table II. The batch size is set as 16. We additionally evaluate LLaMA2 with sequence lengths from 256 to 16K.

Comparisons. We compare our principles optimized dataflow with those from the SOTA dataflow explorer, DAT [15]. DAT supports tensor operator fusion and utilizes mixed-integer programming and genetic algorithms to obtain optimal intra- and inter-dataflow. It has been reported to achieve better dataflow than many other approaches [10], [11], [22].

We apply FuseCU to the TPUv4i [5], and compare it with TPUv4i [5], Gemmini [16], and Planaria [17]. Gemmini supports different stationary in PE, and Planaria utilizes additional hardware to support connection between PE arrays. We also implement a design identical to FuseCU but without tensor fusion, referred to as UnfCU. We summarize these architecture attributes in Table III. All designs undergo our optimization process to select the best dataflow within their supported spaces for fair comparisons.

B. Validating the Optimality of the Four Principles

Fig. 9 shows memory access compared to DAT [15] under different buffer sizes. Two approaches achieve similar memory access, demonstrating that the principles we proposed can yield optimized dataflow. In some cases, our dataflow outperform DAT because DAT uses genetic algorithm that does not guarantee global optimization.

C. Evaluation on FuseCU Architecture

Memory Access. Fig. 10 (bar chart) presents the memory access results normalized to TPUv4i. FuseCU achieves an average reduction in memory access of 63.6%, 62.4%, and 38.7% over TPUv4i, Gemmini, and Planaria, respectively.

This improvement primarily stems from FuseCU's support for all profitable fusion dataflow, enabling it to identify superior solutions. Although still substantial without tensor operator fusion, UnfCU's memory reduction over TPUv4i, Gemmini, and Planaria decreases to 42.6%, 41.0%, and 4.5%, respectively. UnfCU's benefits result from its comprehensive support for the optimal intra-operator dataflow.

Performance. Fig. 10 (line chart) shows performance results, normalized to the target accelerator's peak FLOPs, indicating utilization. FuseCU delivers average performance gains of 1.33×, 1.25×, and 1.14× over TPUv4i, Gemmini, and Planaria, respectively.

This enhancement is driven by flexible stationary dataflow, adaptable tiling, and operator fusion. Models with smaller dimensions benefit from flexible tiling, achieving high utilization with non-square tiling. Flexible stationary strategies enhance utilization when the single stationary is suboptimal. Additionally, fusion further boosts utilization by consolidating small MMs into larger computations.

Sensitivity to Sequence Length. Fig. 11 presents the normalized memory access and utilization across different sequence lengths. FuseCU demonstrates robust performance for both short and long sequences by supporting efficient intra- and inter-operator dataflow, with greater memory access reduction observed for longer sequences.

Area Overheads. Fig. 12 presents the area breakdown across various hardware components within FuseCU when synthesized at 28 nm. This breakdown includes components introduced to support efficient fusion, specifically the XS PE logic, FuseCU resize interconnects, and control units for diverse compute patterns. Other essential elements, such as multipliers, adders, accumulators, base PE registers, control logic, and softmax unit, remain unchanged from a standard systolic array and are therefore not considered overheads.

FuseCU exhibits an area increase of 12.0% over the TPUv4i design. Notably, the FuseCU interconnects and control logic contribute less than 0.1% to this overhead—significantly lower than the 12.6% incurred by Planaria [17], which supports more flexible interconnects.

VI. Conclusion

This paper proposes principle-based dataflow optimization methodology and derives profitable tensor-wise operator fusion. These principles guide the design of FuseCU, the first architecture to unlock the full potential of tensor operator fusion in PEs. Experimental results show that FuseCU significantly reduces memory access and improves hardware utilization with minimal overhead.

REFERENCES

[1] Q. Huang, M. Kang, G. Dinh, T. Norell, A. Kalaiah, J. Demmel, J. Wawrzynek, and Y. S. Shao, "CoSA: Scheduling by Constrained Optimization for Spatial Accelerators," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, Jun. 2021, pp. 554–566.

[2] H. Kwon, P. Chatarasi, V. Sarkar, T. Krishna, M. Pellauer, and A. Parashar, "MAESTRO: A Data-Centric Approach to Understand Reuse, Performance, and Hardware Cost of DNN Mappings," *IEEE Micro*, vol. 40, no. 3, pp. 20–29, May 2020.

[3] S. Zheng, X. Zhang, L. Liu, S. Wei, and S. Yin, "Atomic Dataflow based Graph-Level Workload Orchestration for Scalable DNN Accelerators," in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, Apr. 2022, pp. 475–489.

[4] 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*, ser. ASPLOS '20. New York, NY, USA: Association for Computing Machinery, Mar. 2020, pp. 369–383.

[5] N. P. Jouppi, D. Hyun Yoon, M. Ashcraft, M. Gottscho, T. B. Jablin, G. Kurian, J. Laudon, S. Li, P. Ma, X. Ma, T. Norrie, N. Patil, S. Prasad, C. Young, Z. Zhou, and D. Patterson, "Ten Lessons From Three Generations Shaped Google's TPUv4i : Industrial Product," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. Valencia, Spain: IEEE, Jun. 2021, pp. 1–14.

[6] 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)*, Mar. 2019, pp. 304–315.

[7] S.-C. Kao and T. Krishna, "GAMMA: Automating the HW Mapping of DNN Models on Accelerators via Genetic Algorithm," in *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, Nov. 2020, pp. 1–9.

[8] 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*, ser. ISCA '17. New York, NY, USA: Association for Computing Machinery, Jun. 2017, pp. 1–12.

[9] 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, Jan. 2017.

[10] M. Gao, X. Yang, J. Pu, M. Horowitz, and C. Kozyrakis, "TANGRAM: Optimized Coarse-Grained Dataflow for Scalable NN Accelerators," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '19. New York, NY, USA: Association for Computing Machinery, Apr. 2019, pp. 807–820.

[11] S.-C. Kao, S. Subramanian, G. Agrawal, A. Yazdanbakhsh, and T. Krishna, "FLAT: An Optimized Dataflow for Mitigating Attention Bottlenecks," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ser. ASPLOS 2023. New York, NY, USA: Association for Computing Machinery, Jan. 2023, pp. 295–310.

[12] S. Zheng, S. Chen, P. Song, R. Chen, X. Li, S. Yan, D. Lin, J. Leng, and Y. Liang, "Chimera: An Analytical Optimizing Framework for Effective Compute-intensive Operators Fusion," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, Feb. 2023, pp. 1113–1126.

[13] J. Cai, Y. Wei, Z. Wu, S. Peng, and K. Ma, "Inter-layer Scheduling Space Definition and Exploration for Tiled Accelerators," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, ser. ISCA '23. New York, NY, USA: Association for Computing Machinery, Jun. 2023, pp. 1–17.

[14] S. Zheng, S. Chen, S. Gao, L. Jia, G. Sun, R. Wang, and Y. Liang, "TileFlow: A Framework for Modeling Fusion Dataflow via Tree-based Analysis," in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '23. New York, NY, USA: Association for Computing Machinery, Dec. 2023, pp. 1271–1288.

[15] L. Xu, Z. Mo, Q. Wang, J. Jiang, and N. Jing, "Enabling Multiple Tensor-wise Operator Fusion for Transformer Models on Spatial Accelerators," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, ser. DAC '24. New York, NY, USA: Association for Computing Machinery, Nov. 2024, pp. 1–6.

[16] H. Genc, S. Kim, A. Amid, A. Haj-Ali, V. Iyer, P. Prakash, J. Zhao, D. Grubb, H. Liew, H. Mao, A. Ou, C. Schmidt, S. Steffl, J. Wright, I. Stoica, J. Ragan-Kelley, K. Asanovic, B. Nikolic, and Y. S. Shao, "Gemmini: Enabling Systematic Deep-Learning Architecture Evaluation via Full-Stack Integration," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. San Francisco, CA, USA: IEEE, Dec. 2021, pp. 769–774.

[17] S. Ghodrati, B. H. Ahn, J. Kyung Kim, S. Kinzer, B. R. Yatham, N. Alla, H. Sharma, M. Alian, E. Ebrahimi, N. S. Kim, C. Young, and H. Esmaeilzadeh, "Planaria: Dynamic Architecture Fission for Spatial Multi-Tenant Acceleration of Deep Neural Networks," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. Athens, Greece: IEEE, Oct. 2020, pp. 681–697.

[18] T. Dao, D. Fu, S. Ermon, A. Rudra, and C. Ré, "FlashAttention: Fast and memory-efficient exact attention with IO-Awareness," in *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., vol. 35. Curran Associates, Inc., 2022, pp. 16 344–16 359.

[19] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[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*, ser. MICRO '52. New York, NY, USA: Association for Computing Machinery, Oct. 2019, pp. 754–768.

[21] L. Lu, N. Guan, Y. Wang, L. Jia, Z. Luo, J. Yin, J. Cong, and Y. Liang, "TENET: A framework for modeling tensor dataflow based on relation-centric notation," in *Proceedings of the 48th Annual International Symposium on Computer Architecture*, ser. ISCA '21. Virtual Event, Spain: IEEE Press, Nov. 2021, pp. 720–733.

[22] P. Chatarasi, H. Kwon, A. Parashar, M. Pellauer, T. Krishna, and V. Sarkar, "Marvel: A Data-Centric Approach for Mapping Deep Learning Operators on Spatial Accelerators," *ACM Transactions on Architecture and Code Optimization*, vol. 19, no. 1, pp. 1–26, Mar. 2022.