Enabling Multiple Tensor-wise Operator Fusion for Transformer Models on Spatial Accelerators

Lei Xu, Zhiwen Mo, Qin Wang, Jianfei Jiang, Naifeng Jing*
Shanghai Jiao Tong University, Shanghai, China
xlxl1027@sjtu.edu.cn, sjtuj@sjtu.edu.cn

ABSTRACT

In transformer models, data reuse within an operator is insufficient, which prompts more aggressive multiple tensor-wise operator fusion (*multi-tensor fusion*). Due to the complexity in tensor-wise operator dataflow, conventional fusion techniques often fall short by limited dataflow options and short fusion length. In this study, we first identify three challenges on multi-tensor fusion that result in inferior fusions. Then we propose dataflow adaptive tiling (DAT), a novel inter-operator dataflow to enable an efficient fusion of multiple operators connected in any form and chained in any length. Then, we broaden the dataflow exploration from intra-operator to inter-operator and develop an exploration framework to quickly find the best dataflow on spatial accelerators with given on-chip buffer size. Experiment results show that DAT delivers 2.24× and 1.74× speedup and 35.5% and 15.5% energy savings on average for edge and cloud accelerators, respectively, comparing to the state-of-the-art dataflow explorer FLAT. DAT is open-sourced at https://github.com/lxu28973/DAT.git.

KEYWORDS

dataflow, inter-operator, fusion, accelerator, transformer

1 INTRODUCTION

The proliferation of transformer models has brought a revolution in the realm of machine learning, and the potential of transformer models stems from their fundamental attention mechanism [17]. Fig. 1 visualizes a typical multi-head attention layer, highlighting three primary operator types, i.e. tensor-wise operators (matrix multiplication), row-wise operators (softmax), and point-wise operators (addition), which present distinct compute and memory characteristics. For example, the tensor-wise matrix multiplication (*mm*) operator requires intensive MAC computations particular in $Q \times K^T$ with large sequence length.

For acceleration, various spatial accelerator architectures have been proposed [2, 5, 8, 9]. They often equip a number of processing engines (PEs) and apply specific dataflow to schedule the computation and memory access on these PEs over time and space. For the best performance, prior researches have studied different dataflows

*Naifeng Jing is the corresponding author. This work is supported by the National Natural Science Foundation of China (No. 92264108).

DAC '24, June 23–27, 2024, San Francisco, CA, USA
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0601-1/24/06…$15.00
https://doi.org/10.1145/3649329.3657317

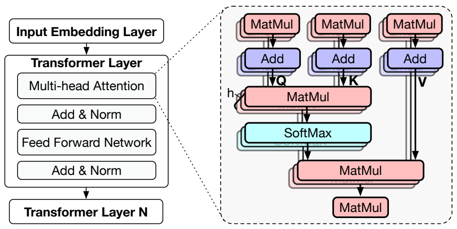


Figure 1: A typical multi-head attention layer in transformer.

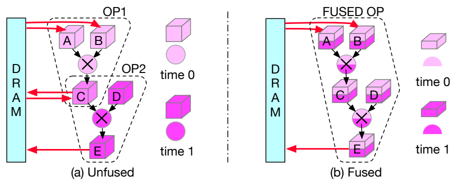


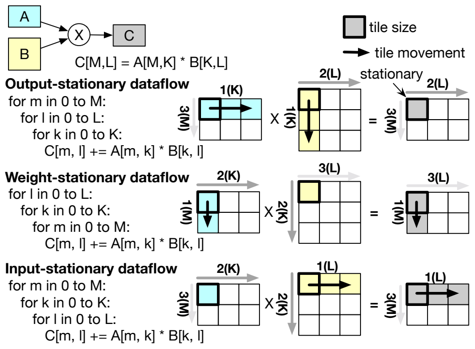
Figure 2: Multi-operator fusion further reduces off-chip memory traffic than unfused operators op_1 and op_2.

to exploit data reuse within a single tensor-wise operator, namely intra-operator dataflow, particular in convolution neural networks (CNNs) [7, 12, 18, 19]. However, $Q \times K^T$, which dominates the computation in transformer when facing long sequence, presents insufficient data reuse because there is no reuse at the output channel and batch dimemsions as in CNNs. Therefore, it poses notable stress on off-chip memory bandwidth as reported in [3, 10, 20].

For optimization, some researches have proposed operator fusion, or inter-operator dataflow [5, 10]. Compared to intra-operator which leaves the intermediate data repeatedly read from and written to memory across an operator boundary as in Fig. 2(a), inter-operator dataflow exploits data reuse between operators. If it can ideally fuse all the operators, the memory traffic on intermediate data can be totally eliminated as in Fig. 2(b). However as in Sec. 3.1, we find that multiple tensor-wise operators fusion *(multi-tensor fusion)* is non-trivial because:

- Contradictive dataflows between two tensor-wise operators prohibit multi-tensor fusion.
- Obstacle operator (e.g. softmax) between two tensor-wise operators prohibits their fusion.
- Directly fusing two compatible tensor-wise operators may not be globally optimal, although each operator bears an optimal intra-operator dataflow.

These challenges confine the conventional fusion techniques to two operators with only a few fixed dataflow options, leaving much design space unexplored for multi-tensor fusion. Our study first addresses these challenges by proposing a dataflow adaptive tiling (DAT). It adapts the tile schedule and size upon fusion, so that any dataflow in a pair of operators can be fused together by eliminating the contradiction or obstacle. Based on DAT fusing two operators, more operators using different dataflows connected in



```
C[M,L] = A[M,K] * B[K,L]
```

Output-stationary dataflow

```
for m in 0 to M:
   for l in 0 to L:
      for k in 0 to K:
         C[m, l] += A[m, k] * B[k, l]
```

Weight-stationary dataflow

```
for l in 0 to L:
   for k in 0 to K:
      for m in 0 to M:
         C[m, l] += A[m, k] * B[k, l]
```

Input-stationary dataflow

```
for m in 0 to M:
   for k in 0 to K:
      for l in 0 to L:
         C[m, l] += A[m, k] * B[k, l]
```

Figure 3: Three intra-operator dataflows 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 OS, "1(K)" implies the innermost loop on dimension K.

different forms can be fused, so that we can examine more dataflow options in the design space for an optimal one. As far as we know, DAT is the first to enable multi-tensor fusion in an arbitrary way rather than previous fusion using fixed dataflow options. And it can fuse other tensor-wise operators not limited to attention layer.

In fact, our proposed DAT opens broader search space for multi-tensor fusion. For fast exploration, we then propose a framework built upon conventional intra-operator dataflow exploration. It decouples the search on scheduling, tiling and mapping, and can analytically find the optimal solution in a faster way.

At last, we apply DAT on attention-based models, including BERT, GPT-2, XLM, Blenderbot, DeBERTa-v2, LLaMA2, and ALBERT, which consists of different sized attention operators. The experiment results show that for typical spatial accelerators on edge and cloud, DAT delivers 2.24× and 1.74× speedup and 35.5% and 15.5% energy savings on average, respectively comparing with state-of-the-art dataflow explorer FLAT [10]. Furthermore, DAT works better on long input sequence (i.e. 64K tokens) without increasing the on-chip buffer size and off-chip memory bandwidth.

2 PREVIOUS WORKS AND BACKGROUND

2.1 Spatial Accelerator

As sketched in Fig. 9, spatial accelerators are primarily structured around processing engines (PEs). Each PE equips a number of MAC units, a local buffer and dedicated interconnections among MACs to facilitate data reuse. In this paper, we assume the most common three-level memory hierarchy, i.e. local buffer, on-chip buffer and off-chip memory as in many spatial accelerators [1, 2, 5, 8, 9, 15].

2.2 Intra- and Inter-Operator Dataflow

Efficiently running DNN and transformer models on spatial accelerator heavily relies on the dataflow, which mainly consists of three key decisions, i.e. scheduling, tiling and mapping. While mapping decides how the MAC units compute the tiled operator and therefore decides the parallelism and data reuse at local buffer level, scheduling and tiling decide how data are reused at the on-chip buffer level. There have been many dataflow studies focusing on convolutions on spatial accelerator [7, 11, 12, 18, 19]. In short, they primarily target on intra-operator dataflow to reduce the off-chip memory traffic with given on-chip buffer size.

Fig. 3 shows three most common intra-operator dataflow options, i.e. output stationary (OS), weight stationary (WS) and input stationary (IS) for an operator $A_{MK} \times B_{KL} = C_{ML}$. First, matrix A, B and C are split into smaller tiles to ensure one tile from each of the three matrices can fit into the on-chip buffer. Regarding the tile scheduling, OS first traverses dimension K as implied by the code innermost loop "1(K)". So C_{ML}, which does not have dimension K, is stationary and reusable. By carefully tuning the tile scheduling and size, spatial accelerator can effectively exploit data reuse to reduce the off-chip memory traffic for a memory-bound operator.

However due to the low operational intensity, exploiting the intra-operator dataflow alone is insufficient in attention layer [3, 10, 20]. As a result, the need of inter-operator dataflow for multi-tensor fusion, arises for keeping intermediate tile data on-chip as much as possible to further reduce off-chip memory traffic. Table 1 briefly summarizes the SOTA dataflow explorers. In short, due to the three challenges, they only fuse point-wise operators [14, 16] or two tensor-wise operators with a fixed dataflow [5, 10]. Later we will see that the lack of holistical search in the dataflow design space will result in local optimization on multi-tensor fusion.

3 DATAFLOW ADAPTIVE TILING ON FUSING TENSOR-WISE OPERATORS

3.1 Three Challenges on Multi-Tensor Fusion

Contradictive dataflow prohibits fusion. Fig. 3 has illustrated the intra-operator dataflow. When we tend to fuse two operators together, the two dataflows chosen by each operator may be contradictive on the binding tensor, because the two dataflows may require the tile in the binding tensor to traverse in different orders, or to be split into different sizes.

Consider two operators mm_1 and mm_2 connecting five tensors calculated in sequence $A_{MK} \times B_{KL} = C_{ML}$ and $C_{ML} \times D_{LN} = E_{MN}$ as in Fig. 4(a). Assume mm_1 and mm_2 adopt IS and OS respectively. Fig. 4(b) shows that the dataflow is contradictive in the binding tensor C_{ML}. To be specific, because dimension L traverses before K with IS in mm_1, the tile ⓒ is calculated as a partial result. While with OS in mm_2, it requires ⓒ as a final result so that dimension L can traverse first. In other words, ⓒ in mm_2 cannot start calculation when mm_1 only gives a partial result. In addition, the tile sizes may also be different. So it is impossible to directly fuse them.

Prior studies tackle this problem by fusing two operators using compatible dataflows. For example, TANGRAM[5] proposes OS-IS fusion with the same tile size as shown in Fig. 4(c). With OS in mm_1, it first traverses dimension K to calculate a tile ⓒ in C_{ML}, which is stationary and serves mm_2 with IS to calculate the partial result tile ⓔ in E_{MN}. Unfortunately, it only works on fusing two operators with compatible dataflow options. When attempting a third operator mm_3, contradiction still arises because a partial result tile ⓔ in mm_2 cannot serve mm_3 to proceed the calculation.

Some other studies proposes other fixed dataflow combinations on fusing two operators, e.g. OS-OS [1, 10]. However, it is obvious that the limited number of fixed dataflow options prohibits the design space exploration for an optimal multi-operator fusion.

Obstacle operator prohibits fusion. Then, we find that the presence of row-wise operators, such as $softmax$ and $reduce$, prevent operator fusion. We call it obstacle operator.

Fig. 4(d) illustrates how $softmax$ acts as an obstacle in the attention layer, where the preceding mm_1 and following mm_2 cannot be

Table 1: Summary of the SOTA dataflow explorers.

Features	Intra-operator	Inter-operator			
	CoSA, ... [7, 12, 18, 19]	DNNFusion, ... [14, 16]	TANGRAM [5]	Flat[10]	DAT (This Work)
Tensor-wise operator fusion scheme	not discussed	do not fuse	OS-IS fusion	OS-OS fusion	adaptive fusion
Inter-operator schedule exploration	×	×	×	×	✓
Tiling size optimization scheme	optimize for intra-operator	optimize for intra-operator	fixed tiling method	fixed tiling method	adaptive tiling method
Row-wise operator fusion scheme	not discussed	do not fuse	not discussed	row-level fusion	tile-level fusion

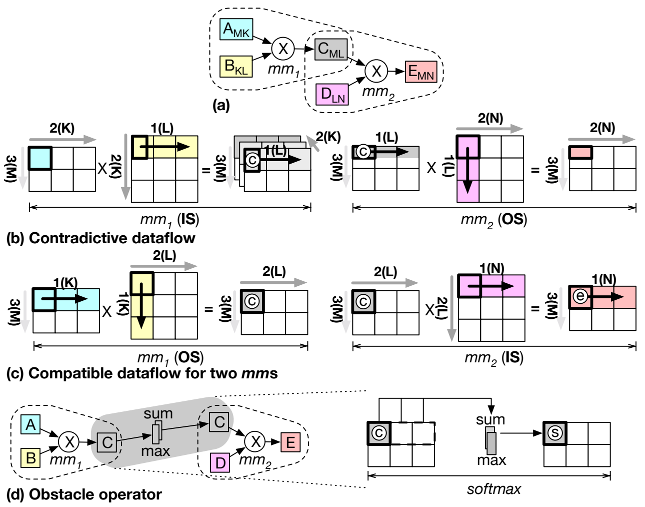


Figure 4: Contradictive dataflow and obstacle operator challenge multi-operator fusion.

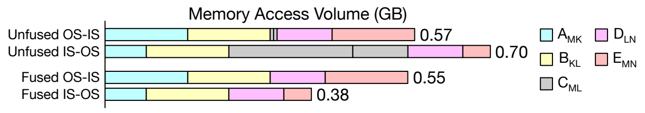


Figure 5: Off-chip memory access volume for unfused OS-IS, IS-OS and fused OS-IS, IS-OS.

fused. This is because with only one tile ⓒ available in C_{ML}, the subsequent operator cannot proceed since $softmax$ algorithm in Eq. (1) needs to compute the maximum and sum of the entire row in C_{ML} to generate one result tile ⓢ. In other words, with only result available in ⓒ, mm_2 cannot proceed to enable fusion.

Directly fusing two operators may not be globally optimal. Even if two operators can be fused together, directly fusing them may not be globally optimal in terms of off-chip memory traffic given the on-chip buffer size. Continuing with the example in Fig. 4(a) using configurations from BERT[4], where mm_1 : $A_{512,768} \times B_{768,64} = C_{512,64}$ and mm_2 : $C_{512,64} \times D_{64,512} = E_{512,512}$. The head number and batch size are set to 12 and 16, and the on-chip buffer size is 512KB. As shown in the statistics in Fig. 5, in the unfused situation, the OS-IS dataflow is preferred. In mm_1, the reading of A_{MK} and B_{KL} requires 0.3GB, and writing to C_{ML} necessitates 0.01GB. In mm_2, reading C_{ML} requires 0.01GB, while accessing D_{LN} and E_{MN} demands 0.25GB. This leads to a total memory access volume of 0.57GB. Upon direct fusion using IS-OS, the total memory access volume decreases to 0.55GB.

When attempting to fuse them using IS-OS with DAT, despite IS-OS requiring 0.7GB (23% more than OS-IS) in their unfused versions, the absence of intermediate data access significantly alters the scenario. Consequently, the fused IS-OS dataflow necessitates only a total volume of 0.38GB, reducing memory traffic by 31% compared to directly fusing the two operators using OS-IS. This reduction could substantially benefit the overall performance in a memory-bound system.

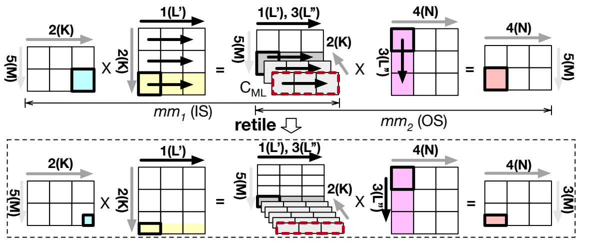


Figure 6: Two steps of DAT, i.e. re-scheduling and re-tiling, to eliminate the contradictive dataflows.

3.2 DAT Eliminating the Contradictive Dataflow

This section will introduce how DAT eliminates the contradictive dataflow to enable multi-tensor fusion with arbitrary dataflows in any length. In short, DAT apply two steps. First, it fuses contradictive dataflows by re-scheduling the contradictive dimensions, and then re-tiling the tiles for efficient buffer usage.

Fig. 6 still uses the example in Fig. 4(b) to illustrate how DAT eliminates the contradiction between mm_1 and mm_2 with IS-OS fusion. Originally as discussed in Sec. 3.1, it is the partial results in dimension L calculated in mm_1 with IS that prevents fusion. So DAT should calculate the final results as soon as possible to enable the fusion. To this goal, DAT traverses dimension L in two ways, i.e. L' for mm_1 and L'' for mm_2, to smooth the dataflow on C_{ML}. The traversal on L' in mm_1 generates partial results, followed by traversal on dimension K in mm_1 for final results in C_{ML}. Then, the traversal on L'' in mm_2 follows traversal on K in mm_1, enabling the final results generated in mm_1 can be reused immediately for mm_2. By this re-scheduling, DAT succeeds in sticking to IS-OS.

Next, to accommodate the intermediate tiles in C_{ML} still in the on-chip buffer, DAT re-tiles the tile shape both in mm_1 and mm_2. Specifically as illustrated at the bottom in Fig. 6, it reduces the tile in both height and width to make the intermediate tiles smaller enough. By a careful re-tiling calculation as in Sec. 4.2, a reduced number of C_{ML} rows are stored on-chip without exceeding the buffer size, and yet the increased accesses on C_{ML} will not result in additional off-chip memory access.

The re-scheduling and re-tiling in DAT also work for other dataflow options, e.g. re-schedule the dimension M and re-tiling in WS-IS fusion. It is important to note that by fusing two operators with arbitrary dataflows, DAT can further extend to more operators connected in any form, and fuse them to an adequate length in terms of memory traffic. By this chaining, DAT enables us to examine all the dataflow combinations rather than a few fixed options for an optimal fusion.

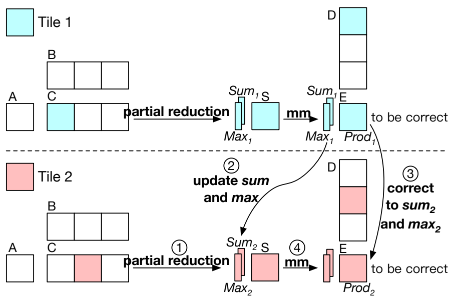


Figure 7: DAT tiles the row-wise operator to enable fusion.

3.3 DAT Eliminating the Obstacle Operator

From the observation in Section 3.1, we find that the key to eliminate the obstacle operator impact is to enable tiling on row-wise operators. In other words, we are able to use a tile of data to start next operator's computation instead of waiting for the whole row.

We use $softmax$ as an example to demonstrate how DAT eliminates the obstacle operator. The $softmax$ is defined in Eq. (1), involving two row-wise operations that compute the maximum ($\text{max} = \text{max}(x_0, ..., x_n)$) and sum ($\text{sum} = e^{x_0-\text{max}} + \cdots + e^{x_n-\text{max}}$).

$$\text{softmax}(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}} = \frac{e^{x_i - x_{\text{max}}}}{\sum_{j=1}^{N} e^{x_j - x_{\text{max}}}} \quad (1)$$

As shown in Fig. 7, DAT employs several steps, i.e. partial computation, update, correction and multiplication to eliminate the obstacle $softmax(C)$ between $A \times B = C$ and $C \times D = E$. Starting with tile 1, DAT computes the partial maximum ($\text{max}_1 = \text{max}(x_0, ..., x_{n1})$) and partial sum ($\text{sum}_1 = e^{x_0-\text{max}_1} + \cdots + e^{x_{n1}-\text{max}_1}$) for the first tile and stores them on-chip. Then, it utilizes these temporary maximum and sum values to compute the $softmax$ and subsequently performs matrix multiplication to obtain the result $prod_1$.

For tile 2, DAT 1) computes the maximum (max_2) and sum (sum_2) for tile 2, 2) updates the max_2 with the old max_1 ($\text{max}_2 = \text{max}(\text{max}_1, \text{max}_2)$) and computes the new sum ($\text{sum}_2 = \text{sum}_1 \times e^{\text{max}_1 - \text{max}_2} + \text{sum}_2$), 3) uses the new max_2 and sum_2 to correct $prod_1$ ($prod_1 = prod_1 \times e^{\text{max}_1 - \text{max}_2} \times \frac{\text{sum}_1}{\text{sum}_2}$), 4) performs the matrix multiplication and add $prod_1$ to obtain $prod_2$.

This process proceeds for the remaining tiles. Eventually when computing the last tile, max_{last} and sum_{last} represent the overall maximum and sum of the entire row, ensuring correctness.

This scheme applies to other row-wise operators such as row-wise $reduction$ by enabling tiling on row-wise operators, therefore enables their fusion with tensor-wise operator. After fusion, it significantly benefits the off-chip memory traffic. Although there is some overhead, including small on-chip buffer size requirement for the *max* and *sum*, and some recomputation on *prod*, their costs are negligible when consider the benefits on memory traffic.

4 DAT EXPLORATION FRAMEWORK

DAT enables all intra-operator dataflow combinations upon multi-tensor fusion. Base on DAT, we broaden the dataflow exploration from intra-operator to inter-operator. For faster exploration, we propose three decoupled optimization steps, i.e. scheduling, tiling, and PE hardware mapping, as illustrated in Fig. 8.

On top of the intra-operator dataflow, which determines the tile scheduling, tile size, and tile mapping within an operator, DAT also considers tile scheduling among multiple operators. To represent inter-operator scheduling, we apply a graph structure where a node denotes an operator and an edge indicates a binding tensor between two operators. For example, in Fig. 4(a), mm_1 and mm_2 are nodes, and the binding tensor C_{ML} forms an edge between them. We introduce a *fusion* attribute on the edge to determine whether the two operators will be fused together. When set to true, the consumer operator (mm_2) should initiate computation as soon as the required tiles in the binding tensor (C_{ML}) are ready. Additionally, we add a *priority* attribute to the node to determine the execution order of operators in the graph when there are multiple executable operators, with higher priority executed first. Regarding intra-operator dataflow annotations, we follow a similar approach to the intra-operator dataflow explorer [12].

We choose an operator, not a tensor, as a node in our graph because DAT may create different dataflows for the binding tensor in two unfused or contradictive operators. As in Fig. 6, the traversal on L' and L'' results in distinct dataflows for the binding tensor C_{ML} in mm_1 and mm_2. This allows for two different tiling on the same tensor. By representing the operator as a node, the binding tensor can have different attributes in two connected operators.

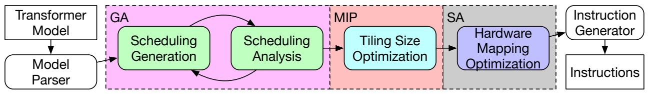


Figure 8: DAT exploration framework with decoupled steps.

4.1 Scheduling Generation and Analysis

Different scheduling of operators heavily impact the DAT results. To examine all the combinations, we apply a genetic algorithm to generate different schedule options and an analytical method to select superior ones. It significantly differs from conventional exploration because it decouples with the tile size optimization (next step), and the analysis is optimized to be fast without requiring time-consuming profiling or simulations used in [16, 19].

Still taking the example in Fig. 4(a), our genetic algorithm works as follows:

1. Initialization. In the DAT graph, we randomly select edges for fusion, randomly assign priorities and dimemsion orders to nodes. Among these assignments, the priorities of nodes are constrained by their dependencies. For example, in Fig. 4(a), we set the edge on tensor C_{ML} as *fusion*, specify that mm_1 should execute before mm_2, and define the dimension order as IS ($1L, 2K, 3M$) for mm_1 and OS ($1L, 2N, 3M$) for mm_2, to initialize a schedule as in Fig. 4(b).

2. Evolution. We introduce mutations in edge *fusion*, node *priority* and dimension order. Crossovers are performed by randomly selecting two individuals and exchanging nodes' intra-operator schedules within parts of their graphs. For example, the individuals in Fig. 4 (b) and (c) are selected, with schedule (IS-OS) and (OS-IS) respectively, and their mm_2 parts are exchanged, resulting two new individuals with schedule (IS-IS) and (OS-OS) respectively.

3. Analyze and Filter. With generated schedules, the memory access volume (MA) and buffer size requirements (BR) can be symbolically analyzed. We extend the intra-operator analysis [12, 18] to inter-operator, and optimize it to make our analysis faster than existing methods. We base on observations: 1) Analyzing MA and BR of a tensor can be done easily by traversing the dimension order instead of analyzing tile movement step by step. 2) The dimensions in other operators would not affect MA in this operator, even if they are fused. 3) Because the tile size in an operator is constant in each execution, liveness analysis for BR computation can be applied at the tensor level instead of the tile. This overlooks tile execution order within an operator, significantly simplifying analysis.

With the symbolic memory access volume (MA) and buffer size requirements (BR), we can filter out inferior or equivalent schedules by comparing them mathematically. For example, in Fig. 4(c), if we mutate the mm_2 dimension order from $(1N, 2L, 3M)$ to $(1N, 2M, 3L)$, although the MA remains the same as before, the BR of C_{ML} changes from $T_L \cdot T_M$ to $L\_size \cdot M\_size$, where T_{dim} represents the tile size of dimension dim, and dim_size represents the overall size of dimension dim. It's evident that $T_L \cdot T_M \leq L\_size \cdot M\_size$, although the specific tile size is not known. So, we filter out the mutated one.

4.2 Tiling Size Optimization

From the previous step, we obtain the generated operator schedules along with their corresponding MA and BR formulas. These are associated with variables representing the tile size T_{dim} and tile number N_{dim} for each dimension:

$$MA = \sum_{tensor,dim_o}^{\substack{unfused\\tensors}} (tensor\_size \times \begin{cases} 1, & \text{tensor is stationary} \\ N_{dim_o}, & \text{tensor is non-stationary} \end{cases}) \quad (2)$$

$$BR = \max(\sum_{tensor}^{\substack{live\\tensors}} ((\prod_{dim_j}^{\substack{buffered\\dims \in tensor}} dim_j\_size) \times \prod_{dim_i}^{\substack{unbuffered\\dims \in tensor}} T_{dim_i})) \quad (3)$$

where dim_o is the dimension in another tensor causing repeated memory access. The $unfused\ tensors$, $stationary$, $live\ tensors$ and $buffered\ dims$ are already determined by the schedule. dim_size is constant. We transform these formulas into a Mixed Integer Programming (MIP) problem to optimize T_{dim}s and N_{dim}s:

$$minimize : MA$$
$$constrains : BR \leq \text{Buffer Size},\ T_{dim} \times N_{dim} \geq dim\_size$$

We employ the Gurobi optimizer [6] to solve the MIP problem and obtain the minimal MA along with the corresponding tile sizes. By comparing the specific MAs among the superior schedules, we select the best one, determining the optimal tile schedule and size.

4.3 PE Hardware Mapping Optimization

The final step maps the tiles to PEs with local buffer to actually do the computation. Firstly, we utilize a flexible partition strategy [5] to divide each tile into smaller workloads, distributing them across the corresponding PEs. Secondly, for each partition scheme, we employ a search-based mapping strategy [12, 18] to explore the optional mapping on the MACs in a PE over space and time.

Note that the mapping at the tile level is decoupled from the prior two steps. The scheduling step will not affect the mapped efficiency, but the tiling step will. So we have to constraint the tile size in the tiling step, e.g. larger than a threshold to ensure a full spatial mapping in a PE. Regarding different PE architectures, this framework also works with their own mapping tools available.

5 EVALUATION AND RESULTS

5.1 Experiment Setup

Spatial architecture parameters. We evaluate DAT on spatial architectures as illustrated in Fig. 9 using two different configurations, i.e. edge and cloud as shown in Table 2 based on [2, 15] for edge and [8, 9] for cloud. Both configurations share the same PE design, which consists of 256 MACs and a 128KB PE buffer.

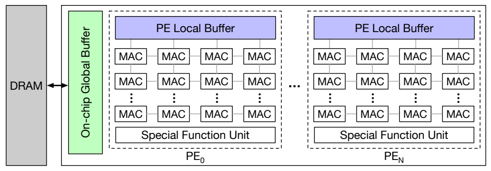


Figure 9: Spatial accelerator architecture modeling.

Table 2: Spatial architecture parameters

Platform	# of PEs	Global Buffer	On-Chip BW	Off-Chip BW
Edge	16	2MB	0.5TB/s	50GB/s
Cloud	512	32MB	4TB/s	400GB/s

Table 3: Transformer model parameters

Model	# of Heads	Seq. Length	Hidden Size
Bert	12	1024	768
GPT-2	12	2048	768
XLM	16	1024	2048
DeBERTa-v2	24	1024	1536
LLaMA2	32	4096 (128-64K)	4096
ALBERT	64	1024	4096

Architecture modeling. We adopt MAESTRO [12] to obtain the execution cycle and power consumption of each PE, which is an open-source model commonly used in dataflow optimization works [11, 13, 19]. For memory modeling, we adopt the methodology in FLAT [10]. Shared memory resources are represented with limited bandwidth, enabling the assessment of memory-bound operations and identification of potential bottlenecks related to memory bandwidth utilization within the accelerator. For the energy parameter used in modeling, the MAC and DRAM energy cost are estimated at 0.02 pJ/op and 7 pJ/bit [19]. The power of SRAMs with different capacities are obtained from commercial SRAM compiler.

Attention-based workloads. We study six recent open-source attention-based models. The model sizes and input sequence lengths are shown in Table 3. The batch size is set as 16. We additionally evaluate LLaMA2 with sequence lengths from 128 to 64K.

Comparisons. We compare DAT with several SOTA dataflow explorers on the given spatial accelerators. For intra-operator, we evaluate [7, 11, 18] and choose the best result in terms of performance for each case. For inter-operator, we choose FLAT [10] and TANGRAM [5], which fuse two operators using fixed dataflow options. We evaluate two DAT versions, i.e. DAT-2 and DAT-∞. DAT-2 restricts at most two operators fused, and therefore shows the potential of considering all the dataflow options on finding the global optimal fusion on two operators. DAT-∞ does not restrict the number of operators to be fused, and therefore shows the capability of DAT on multi-operator fusion.

5.2 Experimental Results

Performance. Fig. 10 reports the performance results normalized to the peak FLOPs of the target accelerator, which also indicates the utilization. For edge, DAT-∞ delivers an average performance improvement of 2.23×, 2.26× and 2.24× over INTRA, TRANGRAM, and FLAT, respectively. For cloud, it is 6.74×, 6.33× and 1.74×.

The improvement mainly comes from that DAT can examine all the possible dataflow options, so it ought to find better solutions.

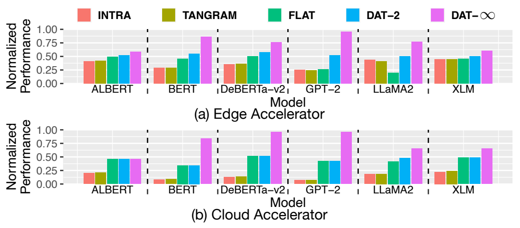


Figure 10: Comparison of normalized performance.

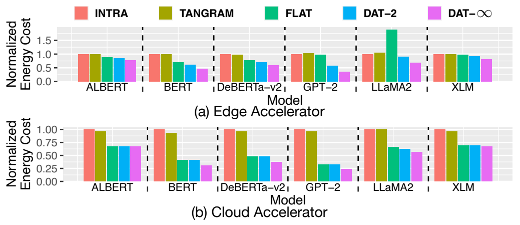


Figure 11: Comparison of normalized energy cost.

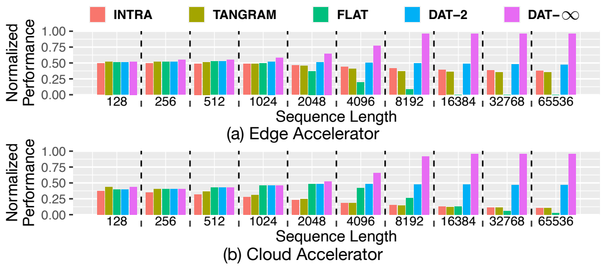


Figure 12: LLaMA2 normalized performance with different sequence length.

Even if two operators are fused, DAT-2's improvement over TRANGRAM and FLAT, i.e. 1.58× and 1.57×(edge), 3.70× and 1.03×(cloud), evidently demonstrates this capability. DAT-∞ enlarges the advantage with 1.43× (edge) and 1.71× (cloud) improvement than DAT-2 by fusing more than two operators, and demonstrates stable improvement even with limited on-chip buffer as in edge accelerator. In contrast, FLAT and TANGRAM applying fixed dataflow fail to fuse more than two operators, or in face of $softmax$.

Energy. Fig. 11 reports the energy cost normalized to INTRA for comparison. For edge, DAT-∞ achieves an average energy reduction of 38.6%, 38.8% and 35.5% compared to INTRA, TRANGRAM and FLAT, respectively. For cloud, it is 53.0%, 51.2% and 15.5%.

Sensitivity to sequence length. Sequence length empowers the attention-based models. The need for long sequences (e.g. 64K) has already gained prominence in the machine learning community [21]. So we vary the sequence length from 128 to 64K in LLaMA2.

Fig. 12 shows that the performance of INTRA and TANGRAM decreases. It is mainly because of the low data reuse in $Q \times K^T$, which dominates the attention computation with longer sequence length. Specifically, FLAT improves at the beginning, but when the sequence becomes longer, its performance drops rapidly because its fusion style stores entire tensor rows on-chip stiffly, causing a substantial buffer requirement. In contrast, DAT-2 and DAT-∞ improves across all sequence lengths. Compared to DAT-2 which experiences slight performance reduction with 64K sequences length, DAT-∞ notably performs better and stable. This largely owns to that it turns the negative long sequence challenge on $Q \times K^T$ into an positive performance opportunity, as larger matrix multiplications can lead to higher data reuse, especially when they are fusible.

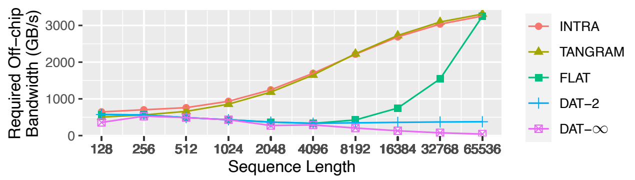


Figure 13: The required off-chip memory bandwidth to efficiently run LLaMA2 on the cloud accelerator.

Minimum memory bandwidth. Fig. 13 reports the off-chip memory bandwidth requirement to achieve high utilization. It shows that INTRA, TRANGRAM and FLAT all demand significant memory bandwidth of thousands of GB/s particulary with longer sequences, which is prohibitive. In contrast, DAT addresses this issue by effectively lowering the requirement by 71.3%, 65.2% and 41.1% against INTRA, TRANGRAM and FLAT, respectively.

6 CONCLUSION

This paper proposes dataflow adaptive tiling (DAT) to enable efficient fusion of multiple operators connected in any form and chained in any length. Based on DAT, we broaden the dataflow exploration from intra-operator to inter-operator and develop a framework to speed up DAT exploration. The experimental results show that DAT outperforms existing dataflow optimizers particularly on long sequence, leading to significant improvements on performance and energy.

REFERENCES

[1] Manoj Alwani et al. 2016. Fused-Layer CNN Accelerators. In *IEEE Micro*.
[2] Yu-Hsin Chen et al. 2019. Eyeriss v2: A Flexible Accelerator for Emerging Deep Neural Networks on Mobile Devices. In *JETCAS*.
[3] Tri Dao et al. 2022. FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness. In *NeurIPS*.
[4] Jacob Devlin et al. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL-HLT*.
[5] Mingyu Gao et al. 2019. TANGRAM: Optimized Coarse-Grained Dataflow for Scalable NN Accelerators. In *ASPLOS*.
[6] Gurobi Optimization, LLC. 2023. Gurobi Optimizer Reference Manual. https://www.gurobi.com
[7] Qijing Huang et al. 2021. CoSA: Scheduling by Constrained Optimization for Spatial Accelerators. In *ISCA*.
[8] Norman P. Jouppi et al. 2017. In-Datacenter Performance Analysis of a Tensor Processing Unit. In *ISCA*.
[9] Norman P. Jouppi et al. 2021. Ten Lessons From Three Generations Shaped Google's TPUv4i : Industrial Product. In *ISCA*.
[10] Sheng-Chun Kao et al. 2023. FLAT: An Optimized Dataflow for Mitigating Attention Bottlenecks. In *ASPLOS*.
[11] Sheng-Chun Kao and Tushar Krishna. 2020. GAMMA: Automating the HW Mapping of DNN Models on Accelerators via Genetic Algorithm. In *ICCAD*.
[12] Hyoukjun Kwon et al. 2020. MAESTRO: A Data-Centric Approach to Understand Reuse, Performance, and Hardware Cost of DNN Mappings. In *IEEE Micro*.
[13] Liqiang Lu et al. 2021. TENET: A Framework for Modeling Tensor Dataflow Based on Relation-Centric Notation. In *ISCA*.
[14] Wei Niu et al. 2021. DNNFusion: Accelerating Deep Neural Networks Execution with Advanced Operator Fusion. In *PLDI*.
[15] Kiran Seshadri et al. 2022. An Evaluation of Edge TPU Accelerators for Convolutional Neural Networks. In *IISWC*.
[16] Yining Shi et al. 2023. Welder: Scheduling Deep Learning Memory Access via Tile-graph. In *OSDI*.
[17] Ashish Vaswani et al. 2017. Attention is all you need. In *NeurIPS*.
[18] Xuan Yang et al. 2020. Interstellar: Using Halide's Scheduling Language to Analyze DNN Accelerators. In *ASPLOS*.
[19] Shixuan Zheng et al. 2022. Atomic Dataflow Based Graph-Level Workload Orchestration for Scalable DNN Accelerators. In *HPCA*.
[20] Size Zheng et al. 2023. Chimera: An Analytical Optimizing Framework for Effective Compute-intensive Operators Fusion. In *HPCA*.
[21] Haoyi Zhou et al. 2021. Informer: Beyond efficient transformer for long sequence time-series forecasting. In *AAAI*.