Data Locality Optimization of Depthwise Separable Convolutions for CNN Inference Accelerators

Hao-Ning Wu and Chih-Tsun Huang
Department of Computer Science
National Tsing Hua University
Hsinchu 30013, Taiwan ROC
wuhoward2002@gmail.com, cthuang@cs.nthu.edu.tw

***Abstract*—This paper presents a novel framework to maximize the data reusability in the depthwise separable convolutional layers with the Scan execution order of the tiled matrix multiplications. In addition, the fusion scheme across layers is proposed to minimize the data transfer of the intermediate activations, improving both the latency and energy consumption from the external memory accesses. The experimental results are validated against DRAMSim2 for the accurate timing and energy estimation. With a 64K-entry on-chip buffer, our approach can achieve the DRAM energy reduction of 67% on MobileNet V2.**

I. Introduction

Deep learning has drawn significant attention recently due to its rapid advance in myriads of intelligent reasoning applications, especially those related to computer vision. The core component that allows deep learning techniques to capture the information (i.e., features) embedded in the input data (e.g., images) is deep convolutional neural network (CNN), where the vast amount of neurons manipluate the features they observe layer by layer.

Many complex CNN models have been developed [1]–[3] to achieve the high accuracy on image classification tasks. More recently, as the demands for real-time intelligent applications grow increasingly, the highly efficient CNN model has become an urgent need on the edge embedded devices with limited resources. One of the essential trends of building up a compact neural network is to adopt depthwise separable convolutions [4]–[6] as the substitutes for standard convolutions in CNNs, which remarkably reduce the number of filter weights.

The highly parallel nature of tensor arithmetics makes CNNs well-suited to be accelerated on specialized hardware. Many works have been proposed [7]–[9] in recent years with different high-performance or energy-efficient architectures. However, even the number of multiply-and–accumulate (MAC) operations are fewer with the introduction of depthwise separable convolutions, the amount of intermediate activations stays the same, which creates strong needs for the high memory bandwidth in addition to the computation capability. Fig. 1 illustrates a typical CNN accelerator consisting of an off-chip DRAM, an on-chip buffer and a group of processing elements (PEs). The computation can only be performed until all required data are moved from the DRAM into the on-chip buffer. Without the proper partition and scheduling, the unnecessary memory accesses could worsen the overall latency and energy consumption.

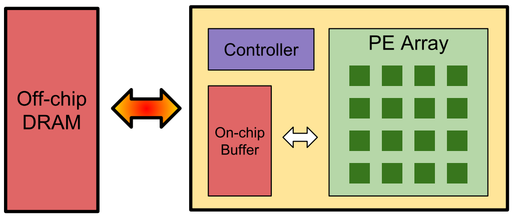


Fig. 1: Block diagram of a CNN accelerator.

In this paper, we propose an optimization framework for depthwise separable convolutions. The optimized tiling sizes along with the execution order can be explored and evaluated under the memory constraint. Our main contributions are summarized as follows:

- An optimized execution order is presented to maximize the data reusability for the tiled matrix multiplications.
- We proposed a novel layer fusion scheme, enabling the optimization of pointwise convolution layers and depthwise separable convolution layers, which can be generalized to standard convolution layers as well.
- We also analyze and categorize the common efficient tiling patterns for the layer fusion, which is valuable for designing the energy-efficient inference accelerators.
- Finally, the proposed approach is validated against DRAMSim2 [10] with the state-of-the-art CNN models for the accurate timing and energy estimation. The experiment result shows that, with a 64K-entry on-chip buffer, the proposed approach can achieve the DRAM energy reduction of 67% on MobileNet V2, and 20% on the decomposed ResNet-50, respectively.

II. Preliminaries

A. Depthwise Separable Convolution

CNN is mainly made up of standard convolutional layers, which perform MAC operations on learnable filters and overlapped input feature maps. For a 3D input with the spatial size of $W_i \times H_i$ and C channels, the M 3D filters of size $W_f \times H_f \times C$ slide over the spatial dimension with the stride U to produce the output feature maps of $W_o \times H_o \times M$.

A depthwise separable convolution performs a depthwise convolution and a 1×1 convolution [6]. Unlike the standard

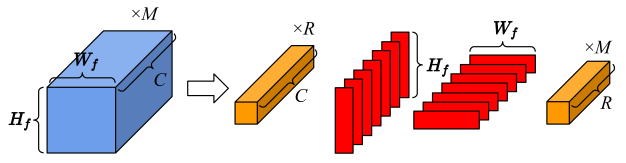


Fig. 2: Rank-R decomposition on 4D filters.

convolution, each channel of input feature maps is convolved with a 2D filter. In addition, the 1×1 convolution, or pointwise convolution, is simply a special case of the standard convolution where $W_f = H_f = 1$. If we reshape the spatial dimension of the input features for the 1×1 convolution to an $L_I \times L_J$ matrix, where $L_I = H_i \times W_i$ and $L_J = C$. Stacking M 1×1 filters horizontally also produces a matrix of the size $L_J \times L_K$ where $L_K = M$. Then the 1×1 convolution is equivalent to a matrix multiplication. Within the context of this paper, we discuss the matrix multiplication to present the proposed approach.

B. CP-Decomposition

Canonical Polyadic (CP) decomposition is a method belonging to the family of the low-rank factorization [11]. It approximates a high-dimensional tensor with the sum of a series of low-rank ones. By decomposing the filters with a user-specific rank R (the blue ones in Fig. 2), a standard convolution layer is decomposed into two 1×1 convolutions (the orange ones) and two depthwise convolutions (the red ones).

Rank selection is crucial to the performance of the decomposed CNNs. The smaller the rank, the lower the computational complexity. However, the model accuracy may degrade to an unacceptable level. The CP decomposition tries to minimize the reconstruction error from the target tensor with algorithms like CP-ALS [12]. In [13], the iterative fine-tuning was proposed to maintain the stability of CP decomposition. Our work adopts the CP-Decomposition to convert the standard convolutions to depthwise separable ones before applying the proposed layer fusion approach.

III. Scheduling for Matrix Multiplication

A. Tiled Matrix Multiplication

Loop tiling is widely used to optimize the data locality by arranging the data access patterns. Fig. 3 shows one of the possible loop nests for the tiled matrix multiplication with two 6x6 matrices and tile sizes of 2. The *load* and *store* functions denote the transfer for a chunk of data between the DRAM and the on-chip buffer. The corresponding tiled matrix multiplication can also be expressed as follows:

$$\begin{pmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{pmatrix} \times \begin{pmatrix} B_{11} & B_{12} & B_{13} \\ B_{21} & B_{22} & B_{23} \\ B_{31} & B_{32} & B_{33} \end{pmatrix} = \begin{pmatrix} C_{11} & C_{12} & C_{13} \\ C_{21} & C_{22} & C_{23} \\ C_{31} & C_{32} & C_{33} \end{pmatrix}, \quad (1)$$

where $\mathbf{A_{ij}}$, $\mathbf{B_{jk}}$ and $\mathbf{C_{ik}}$ are 2x2 submatrices (tiles).

The access function of $\mathbf{C}$ can be promoted to the outer loop because it does not involve with the loop iterator j [9] (see Fig. 3). Therefore, each element in $\mathbf{C}$ needs to be stored only once. Nevertheless, each tile of $\mathbf{A}$ has to be loaded for k times; each tile of $\mathbf{B}$ has to be loaded for i times.

```
for i in range(0, 6, 2):
  for k in range(0, 6, 2):
    initialize(C[i:i+2][k:k+2])
    for j in range(0, 6, 2):
      load(A[i:i+2][j:j+2])
      load(B[j:j+2][k:k+2])
      #on-chip data computation
      for ii in range(i, i+2, 1):
        for jj in range(j, j+2, 1):
          for kk in range(k, k+2, 1):
            C[ii][kk] += A[ii][jj] * B[jj][kk]
    store(C[i:i+2][k:k+2])
```

Fig. 3: Pseudo code of the tiled matrix multiplication.

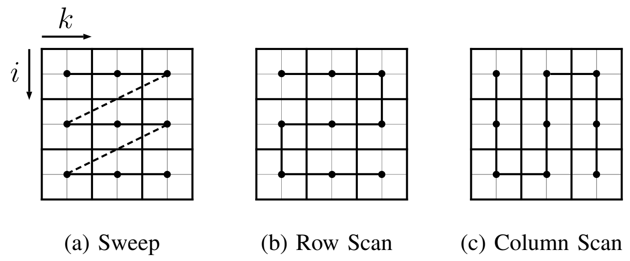


Fig. 4: Trajectory in the promoted matrix $\mathbf{C}$.

Mapping the loop nests in Fig. 3 to realistic accelerators, the three inner loops characterize the on-chip computation to form a *processing pass*. The inner-loop structure depends on the dataflow design. In addition, the three outer loops determine the execution order of loop tiles. For example, the execution order in Fig. 3 is called the *Sweep* order, where the trajectory of tiles in $\mathbf{C}$ is depicted in Fig. 4a. In each processing pass, the accelerator loads the targeted tiles according to the execution order, and then performs the on-chip computation. The total number of DRAM accesses can be estimated by analyzing the arrangement of the outer loops.

B. Cost and Buffer Size Constraint

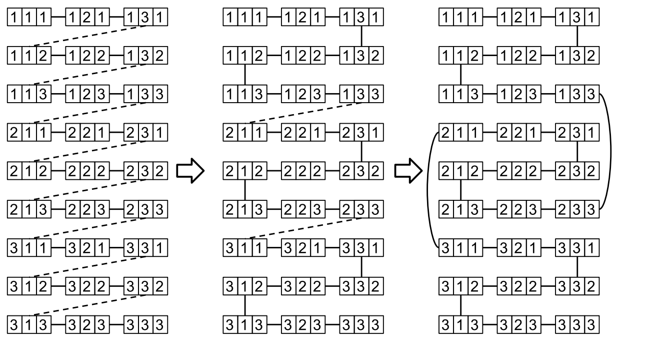
Let $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$ be matrices of sizes $L_I \times L_J$, $L_J \times L_K$ and $L_I \times L_K$, respectively. The tile sizes in each dimension are T_I, T_J, and T_K. Each side of the matrices are divided into $q_I = \lceil L_I/T_I \rceil$, $q_J = \lceil L_J/T_J \rceil$, and $q_K = \lceil L_K/T_K \rceil$ parts, respectively. The tile amount of $\mathbf{A}$, S_A, is therefore $T_I \times T_J$; the tile amount of $\mathbf{B}$ is $S_B = T_J \times T_K$; the tile amount of $\mathbf{C}$, $S_C = T_I \times T_K$. For one processing pass, the on-chip buffer S_{Buffer} has to store one tile from each matrix:

$$0 < S_A + S_B + S_C < S_{Buffer} \quad (2)$$

Rearranging different loop iterators at the innermost loop changes the *promoted matrix*. Table I shows the data transfer amount for the tiled matrix multiplication with *Sweep* order when either $\mathbf{A}$, $\mathbf{B}$ or $\mathbf{C}$ is the promoted matrix. The total number of DRAM accesses to an individual matrix is the product of the *# accesses per element* and the *matrix size*.

TABLE I: Data Transfer Amount for Sweep Order

Promoted Matrix	**Number of Accesses to Each Matrix**		
	A	**B**	**C**
A	$L_I L_J$	$q_I L_J L_K$	$(2q_J - 1)L_I L_K$
B	$q_K L_I L_J$	$L_J L_K$	$(2q_J - 1)L_I L_K$
C	$q_K L_I L_J$	$q_I L_J L_K$	$L_I L_K$



(a) Sweep Order (b) Further Reusing **A** (c) Further Reusing **B**

Fig. 5: The generation of the C-row Scan order.

IV. Proposed Tiling Scheme with Scan Order

A. Maximizing the Reusability with Scan Order

To model the problem of optimizing the execution order for the tiled matrix multiplication, we define the triplet $\boxed{i \mid j \mid k}$ to represent $\mathbf{A_{ij}} \times \mathbf{B_{jk}} = \mathbf{C_{ik}}$, which is an execution instance (or a processing pass) of the tiled matrix multiplication.

Each triplet can be treated as a node on an undirected graph, where the distance between any two nodes are the data reuse amount between them. For example, $\boxed{2 \mid 3 \mid 1}$ and $\boxed{1 \mid 3 \mid 1}$ are identical in the indices j and k, where $j = 3$ and $k = 1$. Therefore, the tile $\mathbf{B_{31}}$ can be reused and the distance between the two nodes is the amount of $\mathbf{B_{31}}$. Our target is to find the optimized execution order of the tiled matrix multiplication, which can be transferred to finding the path with the longest distance to visit every node exactly once. Fig. 5a shows the execution order for the loop nests in Fig. 3. A solid edge indicates there exists a reused tile between the nodes, whereas a dashed edge for no possible reuse. For the original Sweep order, there are 18 reuses for the tiles of **C** between processing passes; no reuse for the rest 8 transitions.

Note that when processing the tiles of different iterator k in **C** (e.g., $\mathbf{C_{11}}$ and $\mathbf{C_{12}}$), the tiles of the same row from **A** are required (e.g., $\mathbf{A_{11}}$, $\mathbf{A_{12}}$, and $\mathbf{A_{13}}$). If the access order is $\mathbf{A_{11}} \rightarrow \mathbf{A_{12}} \rightarrow \mathbf{A_{13}} \rightarrow \mathbf{A_{12}} \rightarrow \mathbf{A_{11}}$, the tile of $\mathbf{A_{13}}$ can be reused for both $\mathbf{C_{11}}$ and $\mathbf{C_{12}}$. The improved order is depicted in Fig. 5b. The same scheme can be further applied to **B** to obtain the final execution order in Fig. 5c, where the execution of any two nodes can reuse a tile. This execution order is called C-row Scan order, where **C** is the promoted matrix and its trajectory is depicted in Fig. 4b. By permuting the three outer loops, there can be six possible execution orders, i.e., A-row, A-column, B-row, B-column, C-row and C-column.

TABLE II: Data Transfer Amount for Scan Order

Execution Order	**Number of Accesses to Each Matrix**		
	A	**B**	**C**
A-row	$L_I L_J$	$q_I L_J L_K$ $-(q_I - 1)T_J T_K$	$(2q_J - 1)L_I L_K$ $-2(q_J - 1)L_I T_K$
A-column	$L_I L_J$	$q_I L_J L_K$ $-(q_I - 1)L_J T_K$	$(2q_J - 1)L_I L_K$ $-2(q_J - 1)T_I T_K$
B-row	$q_K L_I L_J$ $-(q_K - 1)T_I L_J$	$L_J L_K$	$(2q_J - 1)L_I L_K$ $-2(q_J - 1)T_I T_K$
B-column	$q_K L_I L_J$ $-(q_K - 1)T_I T_J$	$L_J L_K$	$(2q_J - 1)L_I L_K$ $-2(q_J - 1)T_I L_K$
C-row	$q_K L_I L_J$ $-(q_K - 1)L_I T_J$	$q_I L_J L_K$ $-(q_I - 1)T_J T_K$	$L_I L_K$
C-column	$q_K L_I L_J$ $-(q_K - 1)T_I T_J$	$q_I L_J L_K$ $-(q_I - 1)T_J L_K$	$L_I L_K$

B. Consideration of Fragmented Tiles

While the width or height of the matrix is not divisible by the tile size, the rightmost or bottommost tile is fragmented and smaller than others. Reusing these *fragmented tiles* makes the solution suboptimal. With a simple modification of Sweep order, the issue can be alleviated.

Typically, the loop iterator counts sequentially from the lower bound to the upper bound (e.g., $\mathbf{A_{11}} \rightarrow \mathbf{A_{12}} \rightarrow \mathbf{A_{13}}$), or the other way around. It forces Scan order to reuse the potential fragmented tiles. In this case, we permute the iterator j in the order of $\mathbf{A_{11}} \rightarrow \mathbf{A_{13}} \rightarrow \mathbf{A_{12}}$ so the last index referenced is $q_J - 1$ instead of q_J. After applying such transformation to the inner iterators j and k of Sweep order, the same patterns of the Scan order generation in Section IV-A can be adopted.

C. Cost and Buffer Size Constraint

Table II compares the number of data accesses with different Scan order schemes, which can be derived by subtracting the reuse data amount from the original accesses in Table I.

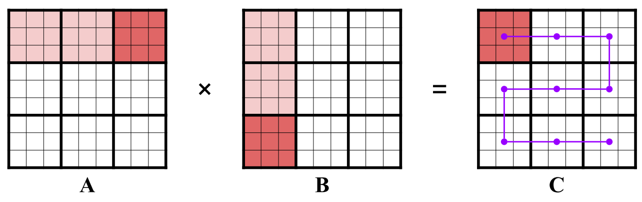
E.g., the C-row Scan order alternately reuses the tiles of **A** in the first and second-last column. For $i = 1$, they are $\mathbf{A_{11}}$ and $\mathbf{A_{1,q_J-1}}$, both with $T_I T_J$ data elements. The reuse occurs whenever the iterator k changes its value, so the data reuse amount from this row is $(q_K - 1)T_I T_J$. Summing up the reuse amount from all rows, we have the total reuse amount of $(q_K - 1)L_I T_J$. In addition, the buffer size constraint is the same as (2).

The Sweep order basically utilizes the *inter-tile data reuse* proposed by [14], where one of the tile sizes among the three matrices is set to one to reduce the search space. However, the proposed Scan order considers possible reuses from every matrix, leading to the further improvement of the data locality with arbitrary tile sizes.

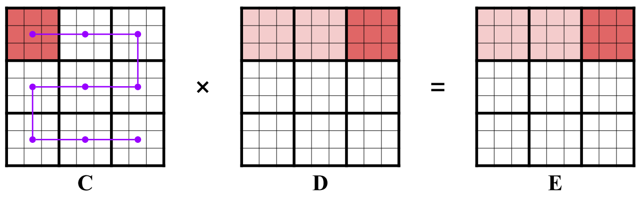
V. Fused Depthwise Separable Convolution

A. Pointwise Layer Fusion

Recently, a specialized accelerator has been presented in [15] to fuse multiple CNN layers. Instead of processing each layer one by one, some intermediate activations can be used to compute the succeeding layer without being stored back to the DRAM. However, only the first few CNN layers are



(a) First 1×1 CONV: $\mathbf{A} \in \mathbb{R}^{L_I \times L_J}, \mathbf{B} \in \mathbb{R}^{L_J \times L_K}, \mathbf{C} \in \mathbb{R}^{L_I \times L_K}$



(b) Second 1×1 CONV: $\mathbf{D} \in \mathbb{R}^{L_K \times L_L}, \mathbf{E} \in \mathbb{R}^{L_I \times L_L}$

Fig. 6: Fusion with (Row-major) Scan order.

considered in [15]. This restriction is a direct result of their fusion scheme, where the fused-layer output is computed with all its required inputs being processed, such that all the partial sums (*psums*) can be summed up in one processing pass, which leads to a considerable amount of on-chip storage.

Our fusion scheme for the 1×1 convolutions breaks through the barrier by allowing psums of the fused-layer output to be accumulated through several processing passes. Fig. 6a shows that the first tile of **C** is computed by summing up 3 tiled matrix multiplications. Before proceeding to the next tile of **C** along the trajectory, the present tile serves as the input to the second layer to be multiplied with the first row of tiles in **D** to get the psums in **E**, as in Fig. 6b. Therefore, the data in **C** never leave the on-chip buffer.

In this scheme, the matrix **C** has to be the promoted matrix, otherwise excessive memory accesses may be incurred. Therefore, the *C Sweep* order in the first layer can be combined with the *A Sweep* order in the second layer to form the *fusion with Sweep* execution order because they access **C** in the same pattern. Similarly, C-row Scan order should pair up with A-row Scan order.

The data transfer amount for the fused 1×1 convolutions can be computed by adding up that of the single layer in previous sections. E.g., because the access to the **C** is completely eliminated, the amount of $2L_IL_J$ can be deducted from the total accesses. The final result is summarized in Table III.

B. Layer Fusion with Depthwise Convolution

A typical CNN may contain different types of layers between the 1×1 convolutions. For the ReLU-like nonlinearities, no extra parameter is needed. For the batch normalization, parameters are shared among all the elements within a channel. While the CNN is deployed in the inference phase, these parameters can be folded into the preceding convolution. Depthwise convolution and pooling perform similar channel-wise computation. Fusing these layers with a preceding 1×1 convolution requires the on-chip buffer allocation for their outputs. While fusing depthwise convolutions, additional storage

TABLE III: Data Transfer Amount for Layer Fusion

Execution Order	Number of Accesses
Fusion with Sweep	$q_K L_I L_J + q_I L_J L_K + q_I L_K L_L + (2q_K - 1) L_I L_L$
Fusion with Scan (Row-major)	$q_K L_I L_J + q_I L_J L_K + q_I L_K L_L + (2q_K - 1) L_I L_L$ $-(q_K - 1) L_I T_J - (q_I - 1) T_J T_K$ $-(q_I - 1) T_K T_L - 2(q_K - 1) L_I T_L$
Fusion with Scan (Column-major)	$q_K L_I L_J + q_I L_J L_K + q_I L_K L_L + (2q_K - 1) L_I L_L$ $-(q_K - 1) T_I T_J - (q_I - 1) T_J L_K$ $-(q_I - 1) L_K T_L - 2(q_K - 1) T_I T_L$

for filters is also needed. In the rest of the paper, we will focus on the fusion of 1×1 convolutions and depthwise convolutions.

The structure of the depthwise convolutions sandwiched between two 1×1 convolutions appeared frequently in recent CNN models such as MobileNets and decomposed models. Fusing the layers within this structure can be illustrated as Fig. 6, except that **C** in Fig. 6b is replaced with **C'** generated from **C** by the depthwise convolution. Due to the sliding window nature of convolutions, a tile of the **C'** will consist of two kinds of data: the complete sums for the second 1×1 convolution and the psums.

During the processing of the fused layers, we iteratively compute the 1st 1×1 convolution to get a tile in **C**, the depthwise convolution to get a tile in **C'**, and then the 2nd 1×1 convolution, and so on. Apart from the data currently needed, the data scheduled to be reused in the future should also be buffered. For example, the psums generated by a depthwise convolution is reused in the forthcoming depthwise convolution so the space for psums (S_{Psum}) need to be allocated in every stage of the computation. Likewise, the storage for the filters (S_{Filter}) and the tiles from the other 1×1 convolutions to be reused should be considered in the same way. As a result, the buffer size constraints can be formulated as follows:

- For the first 1×1 convolution:
$$\begin{aligned} S_{Current} &= S_A + S_B + S_C, \\ S_{Future} &= S_{Psum} + S_{Filter} + \max(S_D, S_E). \end{aligned} \tag{3}$$
- For the intermediate depthwise convolution:
$$\begin{aligned} S_{Current} &= S_C + S_{C'} + S_{Psum} + S_{Filter}, \\ S_{Future} &= \max(S_A, S_B) + \max(S_D, S_E). \end{aligned} \tag{4}$$
- For the second 1×1 convolution:
$$\begin{aligned} S_{Current} &= S_{C'} + S_D + S_E, \\ S_{Future} &= \max(S_A, S_B) + S_{Psum} + S_{Filter}. \end{aligned} \tag{5}$$

For each stage, $0 < S_{Current} + S_{Future} < S_{Buffer}$.

VI. Experimental Evaluation

With the proposed memory access model for different tile execution orders, we adopted branch-and-bound technique to search for the optimized one with both the fewest DRAM accesses and the highest utilization of the on-chip buffer. Our experiment targets at MobileNet V1/V2 and ResNet-50. All the standard convolution layers in ResNet-50 are first decomposed to depthwise seperable convolution layers based on [13]. Our decomposed ResNet-50 retains the top-5 accuracy of 90.51% on ILSVRC2012 [16] dataset by reducing 48% of parameters.

TABLE IV: DRAMSim2 Configuration

System Config		DDR3 SDRAM Config	
Parameter	**Value**	**Parameter**	**Value**
# Channel	1	**tCK**	1.5 ns
Data Bus width	64 bits	**Vdd**	1.5 V
Transaction Queue Depth	32	**# Banks**	8
Command Queue Depth	32	**# Rows**	32768
Row Buffer Policy	Open-page	**# Columns**	1024
Address Mapping Scheme	Scheme6	**Device Width**	8
Using Low Power	True	**Burst Length**	8

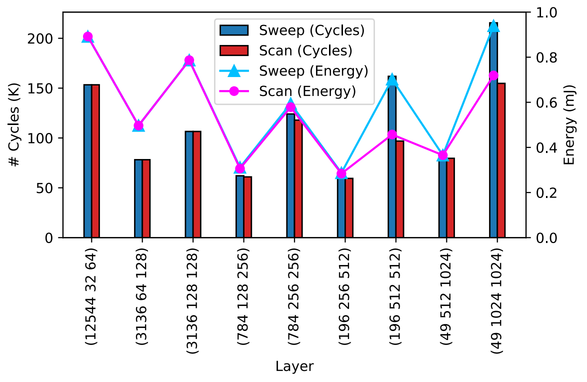


Fig. 7: Simulated cycles & energy of MobileNet V1.

The latency and energy are validated by using DRAMSim2 [10], a cycle accurate trace-based memory system simulator, with the configuration in Table IV. A trace file generator has been implemented, which takes the layer shape, the tile sizes and the data layout as its inputs. The filters are stored in the block data layout for sequential accesses. On the other hand, since the output activations act as the input activations of the succeeding layer, the activations are stored in either row-major or column-major layout to ease the memory access.

A. Optimization of Single Layer

Fig. 7 compares the latency and energy of Sweep and Scan orders for MobileNet V1. Each layer is represented by its matrix shape (L_I, L_J, L_K). From our theoretical computation (see Table I and Table II), the maximum reduction of the DRAM accesses between Scan order and Sweep order is only 2.7%. However, the gap between their simulated results in the layer $(196, 512, 512)$ and layer $(49, 1024, 1024)$ is about 30%. For the layer $(196, 512, 512)$, the tile sizes explored for Sweep order is $(196, 1, 165)$, and $(196, 23, 129)$ for Scan order. The extreme tile sizes destroy the spatial locality and fail to take the advantage of the open-page policy. Accessing to each small strip of data leads to excessive row activation commands to the DRAM, and therefore aggravates the latency and energy consumption of Sweep order a lot.

B. Optimization of the Layer Fusion

The optimized tiling patterns with the fewest DRAM accesses for each layer of MobileNet V1/V2 and decomposed ResNet-50 are analyzed and categorized in Fig. 8, where the depthwise convolutions are skipped in the figure for the simplicity. For patterns I∼III, one or more matrices fully fit into the on-chip buffer, resulting in the optimized number of DRAM accesses (i.e., every element in **A**, **B**, **D** and **E** is accessed once). Interestingly, by sacrificing the tile sizes in **C**, the pattern IV maximizes the data reuse from other matrices. In addition, all other tile size configurations can be categorized as the pattern V.

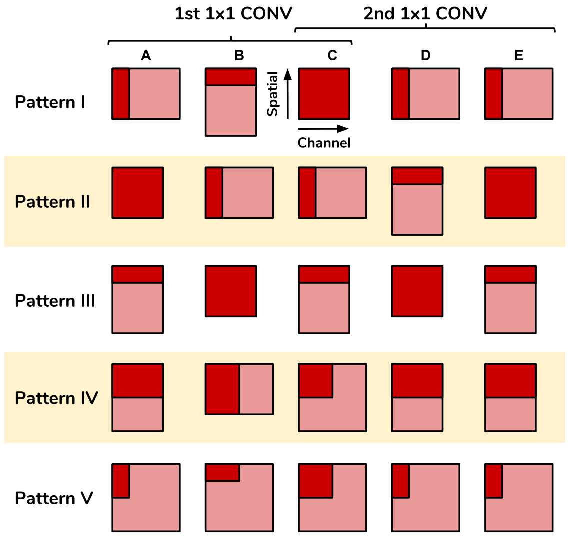


Fig. 8: Common tiling patterns for the optimized fusion.

Our approach can be generalized to the standard convolutions. Taking the pattern III for example, since all the filters are buffered, the fused-layer output can be computed pixel by pixel. For the standard convolutions, the only difference is that the tiles in preceding activation matrices would be taller because the filter window is larger than 1×1. In this case, the pattern III matches the layer fusion scheme of [15]. Similarly, other patterns are also applicable to the standard convolutions, making our approach highly flexible for the layer fusion.

Fig. 9 demonstrates the effectiveness of the layer fusion on MobileNet V2 with a 32K-entry on-chip buffer. The x-axis lists the matrix shapes (L_I, L_J, L_K, L_L), which defines the consecutive 1×1 convolutions with an intermediate depthwise convolution. For every layer, the *Fusion* consistently outperforms *No Fusion* (using Scan order) in both the latency and energy. This significant improvement comes from the unique structure of MobileNet V2, i.e., the inverted residual blocks. So all the layers can take advantage of patterns II∼IV.

C. Comparison on Various Buffer Sizes

Figs. 10 and 11 compare the latency and energy of various buffer sizes among different fusion strategies for the decomposed ResNet-50 and MobileNet V2, respectively. The improvement on MobileNet V2 is substantial because the fusion considers the 1×1, depthwise, and 1×1 convolutions,

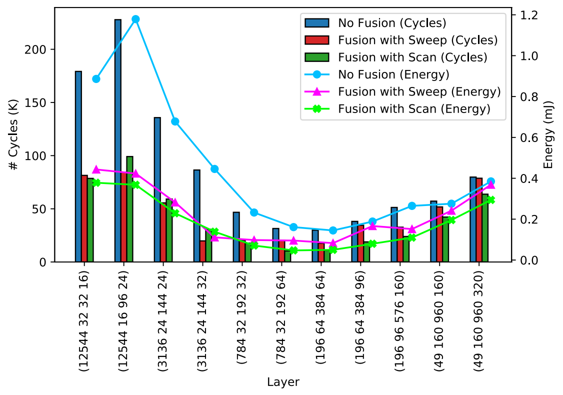
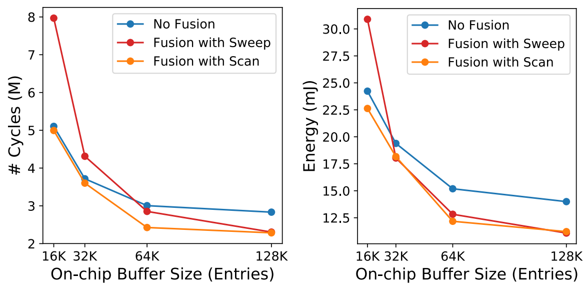
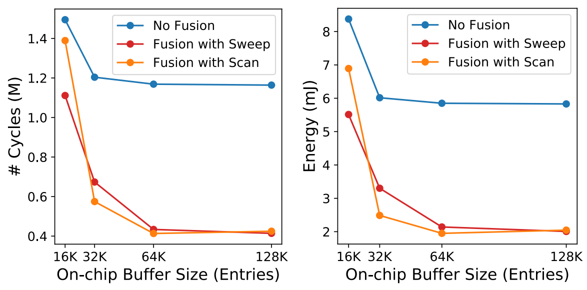


Fig. 9: Simulated cycle & energy of MobileNet V2.



(a) Latency (b) Energy

Fig. 10: Comparison of decomposed ResNet-50.



(a) Latency (b) Energy

Fig. 11: Comparison of MobileNet V2.

instead of only the consecutive 1×1 convolutions in the decomposed ResNet-50. With the on-buffer size of 32K entries, the *fusion with Scan* can reduce 52% latency and 59% energy consumption.

As the buffer size grows, the improvement of *No Fusion* rapidly saturates since one of the matrices can fully fit in the on-chip buffer. On the contrary, the performance of our layer fusion can still increase. With a 64K-entry on-chip buffer, the *fusion with Scan* achieves 20% improvement in both the latency and energy for the decomposed ResNet-50; 67% and 65% for MobileNet V2, respectively. While using an extremely small on-chip buffer, the statistics for *fusion with Scan* or *fusion with Sweep* may skyrocket disproportionately. This is caused by the extreme tile sizes problem mentioned in Section VI-A.

VII. Conclusion

In this paper, we have presented a novel data locality optimization framework for CNN inference accelerators. With the proposed scan execution order of the tiled matrix multiplication, our fusion scheme of depthwise separable convolution layers can minimize the DRAM accesses for the intermediate activations. In addition, efficient tiling patterns are analyzed and categorized. The experimental results with DRAMSim2 show that our approach achieves significant DRAM latency and energy improvement on CNN inference accelerators with limited on-chip buffer.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, 2012, pp. 1097–1105.

[2] C. Szegedy *et al.*, "Going deeper with convolutions," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015, pp. 1–9.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016, pp. 770–778.

[4] A. G. Howard *et al.*, "MobileNets: Efficient convolutional neural networks for mobile vision applications," *ArXiv e-prints*, Apr. 2017.

[5] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," *ArXiv e-prints*, Jan. 2018.

[6] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017, pp. 1800–1807.

[7] Y. Chen *et al.*, "DaDianNao: A machine-learning supercomputer," in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, Dec 2014, pp. 609–622.

[8] Y. 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.

[9] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-based accelerator design for deep convolutional neural networks," in *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2015, pp. 161–170.

[10] P. Rosenfeld, E. Cooper-Balis, and B. Jacob, "DRAMSim2: A cycle accurate memory system simulator," *IEEE Computer Architecture Letters*, vol. 10, no. 1, pp. 16–19, Jan 2011.

[11] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications," *SIAM Rev.*, vol. 51, no. 3, pp. 455–500, Aug. 2009.

[12] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky, "Speeding-up Convolutional Neural Networks Using Fine-tuned CP-Decomposition," *ArXiv e-prints*, Dec. 2014.

[13] M. Astrid and S.-I. Lee, "CP-decomposition with tensor power method for convolutional neural networks compression," in *2017 IEEE International Conference on Big Data and Smart Computing (BigComp)*, Feb 2017, pp. 115–118.

[14] M. Peemen, B. Mesman, and H. Corporaal, "Inter-tile reuse optimization applied to bandwidth constrained embedded accelerators," in *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2015, pp. 169–174.

[15] M. Alwani, H. Chen, M. Ferdman, and P. Milder, "Fused-layer CNN accelerators," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Oct 2016, pp. 1–12.

[16] J. Deng, W. Dong, R. Socher, L. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, June 2009, pp. 248–255.