Mapping Systolic Arrays Onto 3D Circuit Structures: Accelerating Convolutional Neural Network Inference

H. T. Kung
Harvard University
Cambridge, MA, USA
kung@harvard.edu

Bradley McDanel
Harvard University
Cambridge, MA, USA
mcdanel@fas.harvard.edu

Sai Qian Zhang
Harvard University
Cambridge, MA, USA
zhangs@g.harvard.edu

***Abstract*—In recent years, numerous designs have used systolic arrays to accelerate convolutional neural network (CNN) inference. In this work, we demonstrate that we can further speed up CNN inference and lower its power consumption by mapping systolic arrays onto 3D circuit structures as opposed to conventional 2D structures. Specifically, by operating in 3D space, a wide systolic array consisting of a number of subarrays can efficiently implement wide convolutional layers prevalent in state of the art CNNs. Additionally, by accumulating intermediate results along the third dimension, systolic arrays can process partitioned data channels in parallel with reduced data skew for lowered inference latency. We present a building block design using through-silicon vias (TSVs) for the 3D realization of systolic subarrays. We validate the 3D scheme using a 2.5D FPGA design and demonstrate that when mapped onto 3D structures wide systolic arrays can scale up in size without increasing wiring length in interconnecting subarrays. Further, by taking full advantage of 3D structures, we are able to pipeline inference across multiple layers of a CNN over a series of systolic arrays, dramatically reducing the inference time per input sample. These improvements lead to significantly reduced inference latency, which is especially important for real-time applications where it is common to process samples one at a time.**

***Index Terms*—systolic array, convolutional neural network (CNN), deep learning, inference latency, accelerator, 3D-IC implementation, wiring length, power consumption, FPGA**

I. Introduction

Hardware accelerators using systolic arrays have recently been employed with great success for both the training and inference phases of deep convolutional neural networks (CNNs). These designs, such as the Google TPU [1], the ShiDianNao accelerators [2], and numerous other efforts, including [3], [4], [5], [6], [7], have achieved low power consumption and high throughput due to systolic arrays' simplified dataflow architecture and minimum use of I/O (see, e.g., [8], [9]).

In this section, we first provide a brief review of the principle of using systolic arrays for CNNs and introduce terminologies that we will use throughout. Then, we describe the problems we intend to address and preview the contributions of this paper.

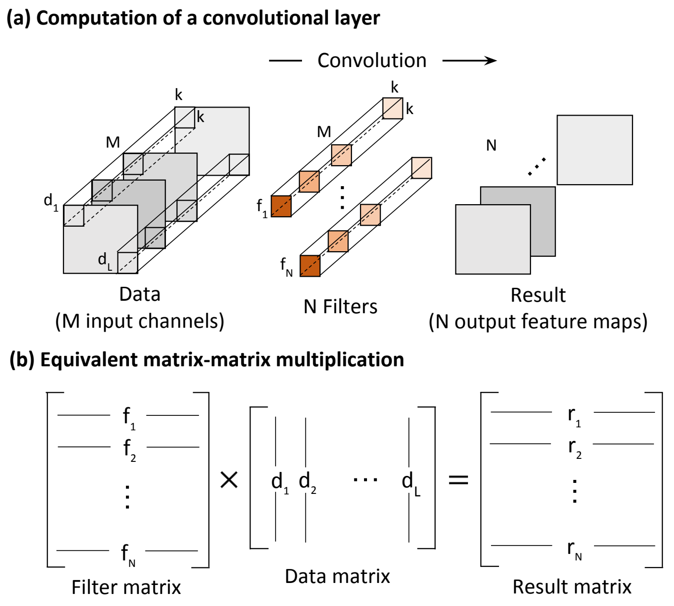


Fig. 1. (a) Computation of a convolutional layer, and (b) viewed as a matrix multiplication.

A. Systolic arrays for Convolutional Layers

It is well known that the bulk of the computation of a convolutional layer for a CNN can be viewed as a matrix-matrix multiplication, or for short, matrix multiplication. Suppose that the convolutional layer has N filters operating on a data volume of depth M, as depicted in Figure 1 (a). Then, the result of the convolution computation is the matrix product of the filter matrix and the data matrix, as depicted in Figure 1 (b) (see a similar illustration in [10] for a pointwise convolution layer with all filters being 1×1).

Figure 2 (a) depicts a systolic array design for this matrix multiplication. It is a *weight-stationary* systolic array in the sense that filter weights stored in the array will not move during computation, whereas input data continuously move bottom-to-top and result data accumulate left-to-right. For systolic array synchronization, items in the data and result

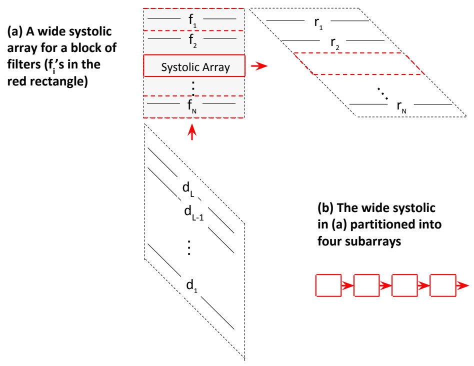

(a) A wide systolic array for a block of filters (f_i's in the red rectangle)
f_1
f_2
Systolic Array
f_N
r_1
r_2
r_N
d_L
d_L-1
d_1
(b) The wide systolic in (a) partitioned into four subarrays

Fig. 2. (a) Partitioning a filter matrix into horizontal stripes, each of which can be implemented with a wide systolic array and (b) partitioning a wide systolic array into subarrays for their implementation on a given fixed-size systolic array.

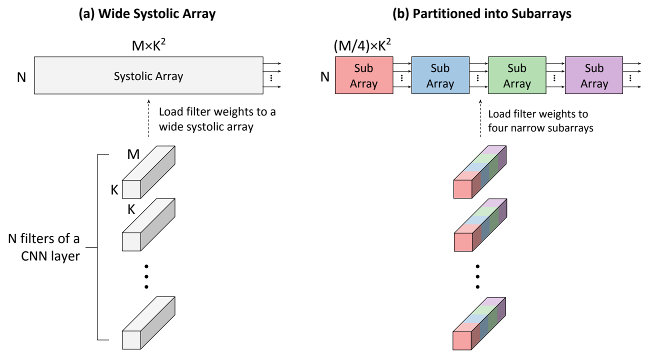

(a) Wide Systolic Array
M×K²
N
Systolic Array
Load filter weights to a wide systolic array
M
K
K
N filters of a CNN layer
(b) Partitioned into Subarrays
(M/4)×K²
N
Sub Array
Sub Array
Sub Array
Sub Array
Load filter weights to four narrow subarrays

Fig. 3. In (a), a single wide systolic array is used to store all of the CNN layer weights. In (b), four narrow systolic arrays (subarrays) divide the weights over the M input channels.

matrices are properly skewed, as shown in the figure. We assume throughout the paper this weight-stationary systolic array design, although other systolic designs may also work in a similar manner, where, for example, input data or result data may be stationary instead.

B. Wide Systolic Arrays

For a given convolutional layer, we use a *wide* systolic array to cover the entire width of the input channels and some subset of features (a block of rows in the feature matrix), as depicted in Figure 2 (a). The *height* of this wide systolic array is the number of these features (rows). These wide systolic arrays are intended to accommodate modern CNNs, such as DenseNet [11] and MobileNet V2 [12], which have layers with 5-20$\times$ more channels than filters. A wide systolic array can process a taller filter matrix by using multiple copies of the array, a single array in multiple passes, or a hybrid method involving both preceding approaches.

For practical implementation on a given fixed-size systolic array hardware (e.g., a chip), we partition a wide systolic array into a series of narrower subarrays so that each of these subarrays can fit the given hardware. Figure 2 (b) illustrates that a wide systolic array in (a) are partitioned into four subarrays.

Figure 3 summarizes the overall approach. A wide systolic array (a) is broken into a series of subarrays, the partition scheme (b) in Figure 3, each of which operating on a subset of the input channels of the large CNN layer. The partial results output from a preceding subarray (*e.g.,* the red array) are used as the initial accumulation values for the following subarray (*e.g.,* the blue array).

C. Mapping a Partitioned Wide Systolic Array Onto 3D Circuit Structures for Efficient Parallel Processing

We are interested in mapping a partitioned wide systolic array onto 3D circuit structures so that subarray modules can run efficiently in parallel. We note that when a wide systolic array scales up to have many subarray modules, the systolic array will need to be folded in order to maintain a reasonable aspect ratio for the layout (such as a ratio close to one). In this case, we will argue that the systolic array can be efficiently mapped into 3D circuit structures which possesses significant advantages over conventional 2D structures (see Figure 5).

For sparse filter matrices, we may pack them first before applying wide systolic arrays. For packing, we may use, for example, *column combining* outlined in an earlier paper [10].

D. Contributions

The main contributions of the paper are summarized as follows:

- **3D Circuit Structures** for partitioning a single wide systolic array into multiple subarrays, each on a different physical layer of a 3D circuit structure. Partial results between physical layers are routed over the third dimension requiring only short wires.
- **Reduced Data Skew through Partition-Accumulation** as depicted in Figure 6 (b). This reduces the input data skew, leading to significant reduction in latency, which is critical in real-time inferences scenarios where samples are processed one at a time.
- A **Systolic Array Slice as a Building Block for 3D Structures** shown in Figure 10, which allows for a uniform design across all physical layers that are connected by through-silicon vias (TSVs).
- **Cross-layer Inference Pipelining** by having multiple CNN layers simultaneously working on a single input sample over a series of systolic arrays one for each layer in a pipelined fashion, as illustrated in Figure 12. Additionally, in Section IV-B, we show that reduced input data skew based on the partition-accumulating scheme leads to more efficient pipelining.
- An **FPGA Implementation** discussed in Section IV-A which uses a 2.5D FPGA for empirical validation of the benefits of the 3D structures over conventional 2D structures for systolic arrays.

II. Background and Motivation

In this section, we describe some properties of systolic arrays and motivate our proposed 3D circuit design presented

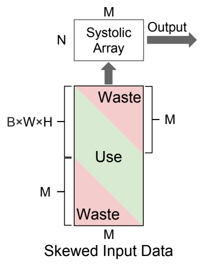


Fig. 4. Skewed input data entering a systolic array. Wasted regions (in pink) are due to the additional skew of one per column. Smaller M lead to higher utilization of the systolic array. Larger B, W, and H (referring to batch size, image width and image height, respectively) also increase utilization.

in Section III.

A. Measuring Systolic Array Utilization

As depicted in Figure 2 (a), systolic arrays require input data to enter at a skewed angle (offset by 1 cycle per column) in order to maintain synchronization of the input and partial results. Figure 4 depicts the skewed input data (the green region) entering the systolic array. The red regions denote unused or wasted area caused by the input skew. As the width of the wide systolic array increases, the data skew also increases, which decreases the systolic array utilization. Conversely, as the input data size (batch size $\times$ image width $\times$ image height) increases, a larger percentage of time is spent performing useful computation, causing the systolic array utilization to increase. This relationship between the width of the systolic array and the size of input data illustrates the issue with wide systolic arrays and processing samples one at time (a batch size of 1), as it can leads to underutilization of the systolic array when the array width is larger the image width $\times$ image height. In Section IV-B, we show that reducing input skew increases systolic array utilization, especially when processing samples one a time.

B. Benefits of 3D Circuit Structures

We have proposed to partition a wide systolic array such as (a) in Figure 3 into multiple subarrays, (b) in the Figure. However, partitioning a larger systolic array into multiple subarrays introduces several challenges when using conventional 2D circuit structures. Figure 5 compares the layout of four subarrays in 2D (a) and 3D (b). The 2D structure requires longer wires to connect systolic arrays that are located on different rows, indicated by the red wires connecting the blue and green subarrays. This leads to longer propagation delay as well as increased power consumption. Additionally, corner turning is required to route the vertical wires from the blue array into the horizontal wires of the green array. The corner turning consumes a routing area proportional to the square of the height of the subarrays. In the 3D design, the third dimension can be used to route the partial results between the physical layers, leading to consistent wire length for all systolic arrays and removing the corner turning issue. In

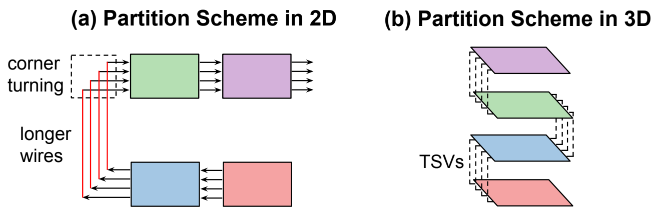


Fig. 5. Using a 2D circuit structure (a), the wire length connecting two subarrays on different rows is proportional to array height, which is highlighted by the longer red wires. Using a 3D circuit structure (b), each subarray is on a different physical layer connected by TSVs, making the wire length independent from the array height. This enables a consistently shorter wire length to be used to connect each pair of subarrays.

Section IV-A, we analyze wire lengths for 2D and 3D layouts using a 2.5D FPGA.

III. Proposed Approach

In this section, we describe two schemes that use multiple subarrays to implement wide CNN layers: a *partition scheme* (P scheme) and a *partition-accumulating scheme* (PA scheme) that reduces the data skew of P scheme through additional accumulators. We then describe how these schemes are mapped onto 3D circuit structures. Additionally, we show how a building block, referred to as a *slice*, can be used to implement these subarrays one on each physical layer of the 3D structure. Finally, we discuss how these 3D structures can be used to perform cross-layer pipelining of CNN inference.

A. Partition and Partition-Accumulating Schemes

Figure 6 presents two partitioning schemes that implement a single wide systolic array. The P scheme in (a) consists of four subarrays, with the output from a previous subarray used as the initial accumulation value for the next subarray. This scheme requires input to be skewed as in a single wide systolic array. The final column of input must be skewed MK^2-1 cycles, where MK^2 is the width of the original wide systolic array.

The PA scheme, (b) in Figure 6, mitigates the input skew in the P scheme by using additional accumulators which add the partial results from each subarray. In this design, the partial results from the previous subarray are not used to initialize the next subarray, which removes the dependency between neighboring subarrays. This reduces the input data skew by a factor of approximately $4\times$ to $(\mathsf{M}/4)\mathsf{K}^2+2$.

Figure 7 provides pseudo-code for systolic array matrix multiplication using the P and PA schemes. For illustration simplicity, the code assumes $\mathsf{K}=1$. This pseudo-code gives a precise definition of the computation performed during matrix multiplication for each scheme. The code for the P scheme is standard matrix multiplication, where the M loop is partitioned across the subarrays. The code for the PA scheme explicitly parallelizes computation over the subarrays and divides the M loop into partitions which are processed by each subarray. The final loop over the number of subarrays corresponds to adding the partial results from each subarray through the additional accumulators. In the future, we plan to use this pseudo-code to

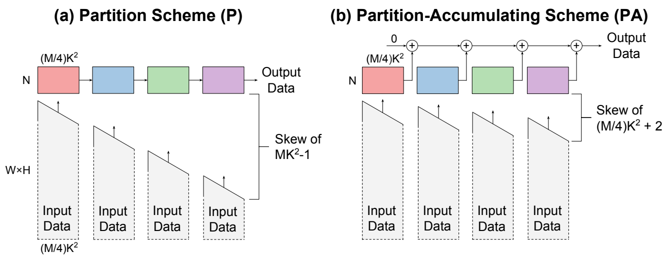


Fig. 6. The P and PA schemes over four subarrays depicted by (a) and (b), respectively. For the P scheme, each column of input data is skewed an additional cycle across all four subarrays, leading to a data skew of MK^2-1 for the final column of the purple subarray. For the PA scheme the input to each additional subarray is offset by only 1 cycle, leading to a data skew of $(\mathsf{M}/4)\mathsf{K}^2+2$, which is an approximately $4\times$ reduction. The element-wise addition for a single row in each subarray is shown at the top of the PA scheme. The dotted lines around the input data denote additional items coming from below.

```
w: CNN layer weights
d: Input data
o: Output data
W: Input data width
H: Input data height
M: number of channels
N: number of filters
a: number of subarrays
so: temporary output
```

(a) Partition Scheme

```
Loop k: 1..W*H
  Loop j: 1..N
    Loop i: 1..M
      o[k,j] = o[k,j] + w[j,i]*d[k,i]
```

(b) Partition-Accumulating Scheme

```
Loop k: 1..W*H
  Loop j: 1..N
    Parallel s: 1..a
      Loop i: 1..(M/a)
        c = s*(M/a) + i
        so[s] = so[s] + w[j,c]*d[k,c]
    Loop s: 1..a
      o[k,j] = o[k,j] + so[s]
```

Fig. 7. Pseudo-code for systolic array matrix multiplication with the partition and partition-accumulating schemes.

generate instructions for each scheme, but currently the code is only used to define the computation.

B. Mapping onto 3D Circuit Structures

We propose to use 3D circuit structures to connect the subarrays of the P and PA schemes. A 3D structure allows for the partial results to be transmitted between subarrays along the third dimension (*i.e.,* through TSVs) which removes wire routing issues when connecting multiple subarrays in conventional 2D circuit structures as discussed in Section II-B.

Figure 8 presents a high-level view of the P and PA schemes mapped onto 3D circuit structures. The P scheme, shown in (a), connects four physical layers, each implementing a subarray, with 3D connections that alternate between the left and right side of each physical layer. This alternating pattern is used to transmit the partial results computed by the previous subarray to the following subarray which then uses the results for accumulation initialization. The layers must alternate in this fashion since the direction of the data changes with each layer. In this example, the red subarray accumulates from right to left and the following blue array accumulates from left to right (as denoted by the arrows).

The PA scheme, shown in (b) in Figure 8, mitigates the input data skew by adding accumulators on one edge of each physical layer, denoted by the darker color regions. These accumulators add a relatively small amount of additional hardware, one adder per row in the subarray, and perform element-wise addition between the rows of the previous and current subarrays. In this design, the partial output from the previous layer is not used to initialize the current layer, and

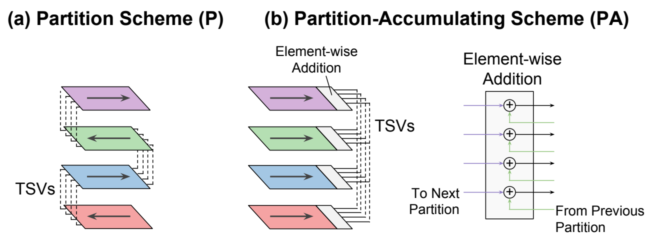


Fig. 8. The partition scheme in (a) connects four physical layers (red, blue, green, and purple) which each implement a subarray. The TSVs, which alternate between the left and right side of each physical layer, are used to transmit partial results to the next layer. The PA scheme in (b) includes an additional adder for every row in each subarray on the right edge of the physical layer. These adders perform element-wise addition between the partial results from the previous array (*e.g.,* the green array) and current array (*e.g.,* the purple array). The direction of dataflow in each subarray is denoted by the black arrow.

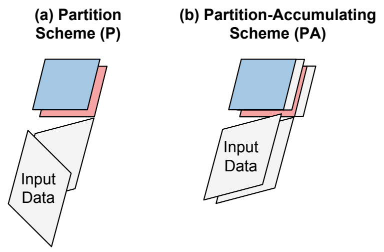


Fig. 9. A 3D view of input data entering two subarrays for the P scheme in (a) and PA scheme in (b). For the P scheme, the data skew for the blue subarray is from left to right in order to match the flow of accumulation from left to right. For the PA scheme, the accumulators are shown on the right side of the two subarrays (dark pink for subarray 1 and dark blue for subarray 2).

therefore all accumulations occur on a single side of each physical layer (all subarrays accumulate from left to right). We provide more details on the building block used to implement each physical layer of the PA scheme in Section III-C.

Figure 9 shows a 3D view of the input data skew for the P and PA schemes. In the P scheme, the orientation of the input data skew alternates between layers based on the direction of accumulation (*e.g.,* from left to right) for that subarray. In the PA scheme, the orientation is the same for all layers and the input data skew is dramatically reduced due to the additional accumulators as discussed in Section III-A.

C. Slice for PA Scheme

Each subarray for the PA scheme is implemented using the slice design shown in (a) in Figure 10. The motivation for the slice is to create a building block which can be duplicated over multiple physical layers in a 3D circuit structure and be used for all convolutional layers in a CNN. The slice takes input, weights and partial results through TSVs. The Demux forwards input and weights to on-slice RAM, which can then be sent to the subarray. The Demux also forwards partial results from the previous slice to the Adders circuit, which performs element-wise addition with the output of the subarray. Normalization and Activation (the N&A circuit) is

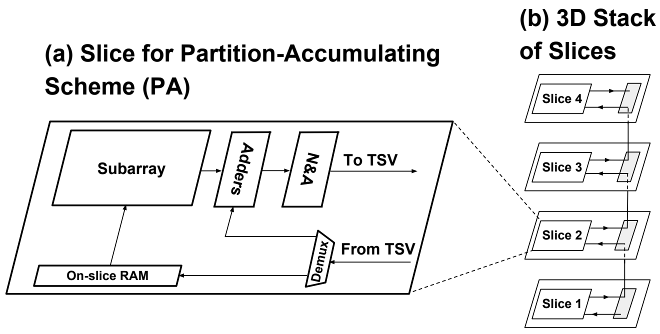


Fig. 10. The slice for the PA scheme (a) receives input data, weights and partial results from the TSV. Element-wise addition is performed in the Adders circuit between the subarray output and partial result from the previous slice. Normalization and Activation is optionally performed in the N&A circuit if the slice is the last partition of a CNN layer. A 3D stack of slices (b) shows how multiple slices are implemented on a 3D circuit (connected by TSVs).

performed only for the final slice in each CNN layer. A 3D stack of slices, (b) in Figure 10, depicts how the uniform slice design can be repeated for each physical layer. The uniform nature of the design highlights the benefits of the simple dataflow architecture of systolic arrays.

D. Cross-layer Pipelining of CNN Inference

While the PA scheme leads to reduced data skew, and thus reduced delay, the benefit could be isolated to only a single CNN layer, as normally the complete output from a convolutional layer is required before starting to process the next convolutional layer. In this section, we propose a method for processing multiple CNN layers in parallel, which dramatically reduces the overall runtime of inference. This is especially important for datasets with large images sizes (*e.g.,* ImageNet [13] with an image size of 224×224), as the number of cycles it takes to process a single CNN layer is proportional to the image size. Bacis *et al.* proposed a dataflow approach for pipelining CNN layers using Iterative Stencil Loops [14]. Here, we show how the dataflow approaches proposed in Section III-A can be used to further improve pipelining and decrease the end to end latency for processing a single sample across the layers of a large CNN.

Figure 11 illustrates how the output from a previous layer (in red) is passed as input into the next layer (in blue). In (a), the output from layer 1 enters as input to layer 2 from below, which follows conventional systolic array design. In (b), layer 2 has been transposed so that the input enters from the right side. Corner turning is required to route the input to layer 2. In (c), a 3D view of (b) is shown, with the corner turning replaced by local connections over the third dimension. We utilize this 3D approach to implement multiple CNN layers without requiring corner turning regions to connect the layers.

Figure 12 shows a cross-layer pipeline consisting of three convolutional layers. Input is passed into the first layer (the red rectangle) and output from the layer will start arriving after a number of cycles corresponding to the subarray width. This intermediate output is immediately passed into the second layer and the pipelining pattern is repeated for each additional

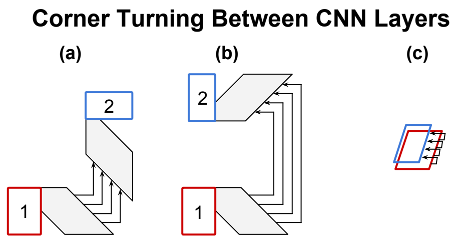


Fig. 11. Multiple orientations of output from a previous layer (layer 1 in red) as input to the following layer (layer 2 in blue). In (c) under a 3D circuit structure, the corner turning shown in (b) is avoided by using connections along the third dimension.

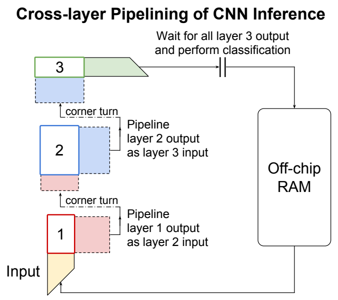

Cross-layer Pipelining of CNN Inference
Wait for all layer 3 output and perform classification
3
corner turn
Pipeline layer 2 output as layer 3 input
2
corner turn
Pipeline layer 1 output as layer 2 input
1
Input
Off-chip RAM

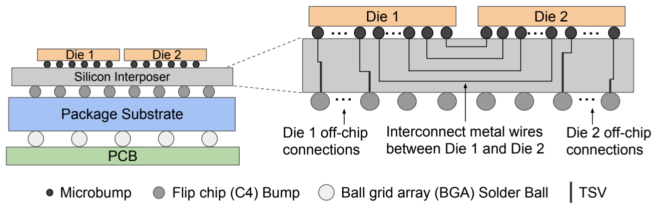
Fig. 12. Cross-layer pipelining for a CNN with 3 convolutional layers (the red, blue, and green rectangles denoted 1, 2, and 3 respectively). Each CNN layer is implemented as a 3D stack of slices as shown in (b) in Figure 10. The dashed colored rectangles represent intermediate output from a layer, which is pipelined as input to the following layer. The corner turns designate the use of 3D connections as shown in (c) in Figure 11. For a given input sample, a barrier (the black vertical bars) is present before classification as it requires all output from layer 3. Input is read from Off-chip RAM into the pipeline. Predictions are written back to Off-chip RAM.

layer, up to the final layer, which requires all output before making a prediction (in the case of a classification task).

Cross-layer pipelining decouples the size of the input data (*i.e.,* the image width and height) from the delay associated with a single systolic array processing the input. For cases where delay is an important factor, such as a real-time scenarios with a batch size of 1, this technique leads to a dramatic reduction in the number of cycles required to process a single sample. In Section IV-B, we show that cross-layer pipelining can lead to a $10\times$ reduction in latency for inference for deep CNNs with over 100 convolutional layers. Generally, the larger the image size, the bigger the advantage for cross layer pipelining, as the delay associated with image size is mitigated.

IV. Evaluation

In this section, we evaluate the proposed benefits of 3D circuit structures for the P and PA schemes and cross-layer pipelining as outlined in Section III. Section IV-A compares 2D and 3D systolic array designs as depicted in Figure 5 implemented on a 2.5D FPGA. Section IV-B provides simulation results for the runtime of inference using of the P and PA schemes with and without cross-layer pipelining.



Die 1
Die 2
Silicon Interposer
Package Substrate
PCB
Die 1 off-chip connections
Interconnect metal wires between Die 1 and Die 2
Die 2 off-chip connections
Microbump
Flip chip (C4) Bump
Ball grid array (BGA) Solder Ball
TSV

Fig. 13. The side view of the 2.5D FPGA.

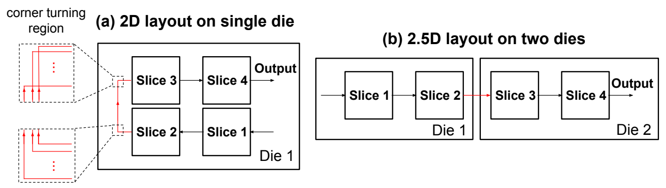


Fig. 14. The 2D layout of the P scheme (a) and the 2.5D layout of the P scheme (b). The interconnect metal wires between Slice 2 and Slice 3 are highlighted in red for both layouts. The dashed square regions in (a) denotes the area taken by corner turning. All wire lines represent 50 1-bit metal wires.

A. FPGA Implementation of 3D Circuit Structures

For the FPGA implementation of the P scheme, we use the Xilinx XC7V2000TFLG1925 chip [15]. The chip size is 23.85mm x 21.65mm [16] and the total amount of available on-chip hardware resources is summarized as follows: 1,221,600 LUT (LookUp Table) elements, 2,443,200 Flipflops, 1,292 on-chip RAMs with 36Kb per on-chip RAM.

The chip contains two dies which are connected by approximately ten thousand interconnect metal wires in the silicon interposer. This architecture is referred to as 2.5D FPGA. Figure 13 shows the side view of the dies, silicon interposer, package substrate and PCB (printed circuit board). The TSVs provide connections for the power/ground, clocking, memory and control signals between the off-chip components and the dies.

We design and synthesize the P scheme using the Xilinx Vivado Design Suite (2018.1). We implement the P scheme for four slices on the FPGA running at a clock frequency of 150MHz. Each slice contains a 50 by 50 subarray. The subarray is implemented in the bit-serial fashion such that one bit of the result will be generated at each row of the subarray per clock cycle. Figure 14 (a) shows the 2D layout of the P scheme, with the four slices placed in two rows. Figure 14 (b) shows the 2.5D layout of the P scheme, where Slice 1 and Slice 2 are placed on Die 1 and Slice 3 and Slice 4 are placed on Die 2. In the 2.5D design, the four slices are placed in a row with Slice 2 and Slice 3 connected by the shortest 50 interconnect metal wires between Die 1 and Die 2.

We measure the length of the longest interconnect metal wire between Slice 2 and Slice 3, which are on different rows in the 2D layout. Among these 50 interconnect metal wires, highlighted in red in Figure 14 (a), the length of the longest wire is 6.35mm. Moreover, the length of the longest wire grows in proportion to the height of the systolic array since the interconnect wires between Slice 2 and Slice 3 need to span the height of the slices. In the 2.5D design, the length of the longest interconnect wire between Slice 2 and Slice 3 is only 2.11mm since the interconnect metal wires between Slice 2 and Slice 3 do not have to span the height of the slices as in the 2D design. For a 50 by 50 systolic array, this leads to the 2.5D design achieving a $3\times$ reduction in the longest wire length and therefore a $3\times$ reduction in the propagation delay over the 2D design. In addition, there is a significant gain in power consumption.

We also measure the total area occupied by the two sets of corner turning wires connecting Slice 2 and Slice 3 in the 2D design. This corner turning area is shown by the dashed square regions of (a) in Figure 14. The width of each region is the pitch of the 50 interconnect metal wires, corresponding to the 50 rows in each subarray. Therefore the area of the corner turning regions grow as more rows are added to subarrays. The total area of the corner turning regions is 4.52mm^2, which is approximately 1% of the total chip area. In the 2.5D design, all the corner turning wires connecting Slice 2 and Slice 3 are inside the silicon interposer.

B. Simulation of PA Scheme and Cross-layer Pipelining

In order to show the benefits of the proposed PA scheme and cross-layer pipelining, we provide simulation results over a range of CNN widths (*i.e.,* number of filters per layer) and CNN depths (*i.e.,* number of layers) across both the P scheme and PA scheme with and without cross-layer pipelining. For all experiments, we use MobileNet V2 [12], which employs a bottleneck architecture that alternates between a convolutional layer with many filters and fewer channels (tall and skinny) and fewer filters and many channels (short and wide). All layers are pointwise convolution (*i.e.,* the filters are 1×1), and a shift operation as described in [17] is employed between each pair of layers in place of separable convolution used in MobileNet V2. The input to the network is the CIFAR-10 dataset [18], which consists of 32×32 RGB images. Each shift operation increases the pipeline delay by a row of pixels (*e.g.,* 32 cycles for 32 pixels). A single fully connected layer is used at the end of the network and is 2-5% of the total number of cycles for inference. The delay added by the shift operations and the final fully connected layer are accounted for in all simulation results. For all networks, we assume that there are sufficient hardware resources to implement all layers simultaneously. Additionally, each CNN layer is partitioned into four subarrays, as depicted in Figure 6, for both the P and PA schemes.

The number of filters in each layer is parameterized by a width multiplier, as discussed in [12], which is simply multiplied by the baseline number of filters for a layer. A larger width multiplier leads to a wider network, as layers will have more filters and channels.

Figure 15 (a) reported the simulated number of cycles required for inference over a range of CNNs as the width multiplier is increased from 0.1 to 2, increasing the number of CNN weights from 15K to 1.5M. The depth of all networks is fixed to 20 layers. The PA scheme provides a $1.8\times$ reduction

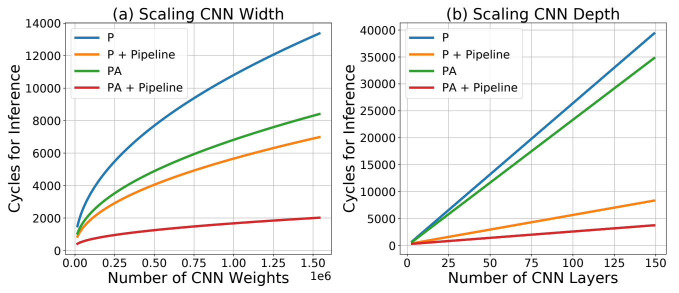


Fig. 15. Number of cycles to process a single sample using the P and PA schemes with and without cross-layer pipelining as the width of the CNN increases in (a) and the depth of the CNN increases in (b). In (a), the number of CNN weights is increased by adding additional filters to each layer and each CNN has 20 convolutional layers. In (b), the width of the network remains constants regardless of the number of layers, alternating between a layer with 50 filters and 200 channels and a layer with 200 filters and 50 channels (*i.e.,* the bottleneck architecture of [12]).

in number of cycles over the P scheme for the large CNN with 1.5M weights. Since the input data skew increases as the width of each layer grows, the reduced skew of the PA scheme allows layers to finish processing more quickly. Adding cross-layer pipelining further reduces the number of cycles by a factor of $2\times$ for the P scheme and a factor of $4\times$ for the PA scheme. The large improvement for the PA scheme is due a $4\times$ reduction is data skew, which allows for the next layer to start processing up to $4\times$ sooner.

Figure 15 (b) shows the number of cycles required for inference as the number of layers in the CNN is increased while keeping the size of each layer fixed. In this case, we see that the schemes with pipelining dramatically outperform the schemes without pipelining. Specifically, the PA scheme with pipelining reduces the number of cycles by $10\times$ over the PA scheme without pipelining when 150 convolutional layers are used. This improvement is due to many layers being able to work in parallel with cross-layer pipelining (as shown in Figure 12) as opposed to only a single layer working at a time without pipelining. This highlights the importance of cross-layer pipelining for deep networks, as the added depth which conventionally has a significant impact on training and inference runtime is mostly mitigated.

V. Conclusion

In this work, we introduce two partitioning methods for systolic arrays, the P and PA schemes, in order to support wide convolutional layers common in state of the art CNNs. We demonstrate that the PA scheme reduces the input data skew by a factor proportional to the number of partitions used, leading to a significant reduction in inference runtime as shown in Section IV-B. We claim that these partitioning schemes are naturally supported by 3D circuit structures, as partial results between subarrays on different physical layers can be routed along the third dimension, leading to a consistent short wire length. In Section IV-A, we support this claim with empirical results on wire length for 2D and 2.5D layouts of systolic arrays implemented on a 2.5D FPGA. Additionally, we demonstrate the effectiveness of cross-layer pipelining when each layer is implemented using the PA scheme, leading to a 10x reduction in inference runtime over the baseline P scheme as shown in Figure 15. We hope that these results encourage further work to explore the landscape of efficient systolic array designs in conjunction with pipelining computation over entire CNNs as well as 2.5D and 3D implementations for both training and inference.

Acknowledgements

This work is supported in part by a gift from MediaTek USA and in part by a Joint Development Project with TSMC.

References

[1] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*. ACM, 2017, pp. 1–12.

[2] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam, "Shidiannao: Shifting vision processing closer to the sensor," in *ACM SIGARCH Computer Architecture News*, vol. 43, no. 3. ACM, 2015, pp. 92–104.

[3] R. Merritt, "Arm at risk on ai chip marketc," *EE Times India*, April 2018.

[4] 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, 2017.

[5] S. Wang, D. Zhou, X. Han, and T. Yoshimura, "Chain-nn: An energy-efficient 1d chain architecture for accelerating deep convolutional neural networks," in *2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2017, pp. 1032–1037.

[6] S. Zhang, Z. Du, L. Zhang, H. Lan, S. Liu, L. Li, Q. Guo, T. Chen, and Y. Chen, "Cambricon-x: An accelerator for sparse neural networks," in *Microarchitecture (MICRO), 2016 49th Annual IEEE/ACM International Symposium on*. IEEE, 2016, pp. 1–12.

[7] X. Wei, C. H. Yu, P. Zhang, Y. Chen, Y. Wang, H. Hu, Y. Liang, and J. Cong, "Automated systolic array architecture synthesis for high throughput cnn inference on fpgas," in *Design Automation Conference (DAC), 2017 54th ACM/EDAC/IEEE*. IEEE, 2017, pp. 1–6.

[8] H. T. Kung and C. E. Leiserson, "Systolic arrays (for vlsi)," in *Sparse Matrix Proceedings 1978*. Society for Industrial and Applied Mathematics, 1979, pp. 256–282.

[9] H. T. Kung, "Why systolic architectures?" *IEEE Computer*, vol. 15, pp. 37–46, 1982.

[10] H. Kung, B. Bradley, and Q. Z. Zhang, "Adaptive systolic arrays for sparse convolutional neural networks," in *Pattern Recognition (ICPR), 2018 24th International Conference on*. IEEE, 2018.

[11] G. Huang and Z. Liu, "Densely connected convolutional networks."

[12] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation," *arXiv preprint arXiv:1801.04381*, 2018.

[13] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR09*, 2009.

[14] S. Bang, J. Wang, Z. Li, C. Gao, Y. Kim, Q. Dong, Y.-P. Chen, L. Fick, X. Sun, R. Dreslinski *et al.*, "14.7 a 288μw programmable deep-learning processor with 270kb on-chip weight storage using non-uniform memory hierarchy for mobile intelligence," in *Solid-State Circuits Conference (ISSCC), 2017 IEEE International*. IEEE, 2017, pp. 250–251.

[15] "7-series product selection guide," https://www.xilinx.com/support/documentation/selection-guides/7-series-product-selection-guide.pdf.

[16] "Device reliability report," https://www.xilinx.com/support/documentation/user_guides/ug116.pdf.

[17] B. Wu, A. Wan, X. Yue, P. Jin, S. Zhao, N. Golmant, A. Gholaminejad, J. Gonzalez, and K. Keutzer, "Shift: A zero flop, zero parameter alternative to spatial convolutions," *arXiv preprint arXiv:1711.08141*, 2017.

[18] A. Krizhevsky, V. Nair, and G. Hinton, "The cifar-10 dataset," 2014.