Maestro: A Memory-on-Logic Architecture for Coordinated Parallel Use of Many Systolic Arrays

H. T. Kung*, Bradley McDanel*, Sai Qian Zhang*, Xin Dong*, Chih Chiang Chen†
*Harvard University
kung@harvard.edu, mcdanel@fas.harvard.edu, {zhangs, xindong}@g.harvard.edu
†MediaTek
john-cc.chen@mediatek.com

Abstract*—We present the Maestro memory-on-logic 3D-IC architecture for coordinated parallel use of a plurality of systolic arrays (SAs) in performing deep neural network (DNN) inference. Maestro reduces under-utilization common for a single large SA by allowing parallel use of many smaller SAs on DNN weight matrices of varying shapes and sizes. In order to buffer immediate results in memory blocks (MBs) and provide coordinated high-bandwidth communication between SAs and MBs in transferring weights and results Maestro employs three innovations. (1) An SA on the logic die can access its corresponding MB on the memory die in short distance using 3D-IC interconnects, (2) through an efficient switch based on H-trees, an SA can access any MB with low latency, and (3) the switch can combine partial results from SAs in an elementwise fashion before writing back to a destination MB. We describe the Maestro architecture, including a circuit and layout design, detail scheduling of the switch, analyze system performance for real-time inference applications using input with batch size equal to one, and showcase applications for deep learning inference, with ShiftNet for computer vision and recent Transformer models for natural language processing. For the same total number of systolic cells, Maestro, with multiple smaller SAs, leads to 16x and 12x latency improvements over a single large SA on ShiftNet and Transformer, respectively. Compared to a floating-point GPU implementation of ShiftNet and Transform, a baseline Maestro system with 4,096 SAs (each with 8x8 systolic cells) provides significant latency improvements of 30x and 47x, respectively.

***Index Terms*—systolic arrays, memory-on-logic 3D-IC, computer architecture, combining switch, deep neural network, convolutional neural network, Transformer**

I. Introduction

In recent years, the success of deep learning has spanned many fields, including manufacturing, finance, and medicine. Due to this success, a new focus has been placed on application-specific deep learning processor arrays for efficient DNN inference on cloud, edge, and end devices. It is known that systolic arrays (SAs) can be effective for this purpose, as demonstrated by systolic array matrix multiplier units in the Google TPU [6].

Real-world DNN workloads consist of matrix multiplications with learned weight matrices of various shapes and sizes. However, a single large SA is underutilized when processing a smaller weight matrix, as it has more systolic cells than weights in the matrix, meaning some cells will be turned off. In this case, instead of a single large SA, a collection of many smaller SAs could be used; these small SAs can work independently to process small matrices with high utilization and also work in a coordinated fashion to process large matrices. Yet, using multiple smaller SAs leads to additional communication requirements, as the system must operate on intermediate or partial results computed by SAs. This requires the SAs to have high-bandwidth and flexible parallel access to multiple memory blocks (MBs).

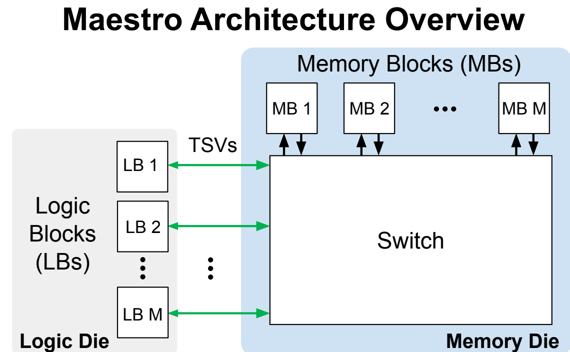


Fig. 1: Using memory-on-logic 3D-IC package technology, Maestro connects memory blocks (MBs) to logic blocks (LBs) each holding a systolic array (SA) through a switch.

To address this SA-MB communication requirement, we present Maestro, a novel memory-on-logic 3D-IC architecture, which can scale up along the horizontal plane with shortened wires in connecting SAs to MBs by utilizing vertical 3D-IC interconnects. In the post-Moore's Law era, where higher computing bandwidth can only be achieved by increasing chip area rather than reducing device size, this horizontal scaling ability is critically important.

Figure 1 provides an overview of the Maestro architecture, which connects logic blocks (LBs), each containing an SA, on a logic die to MBs, each being a memory bank, on a memory die through a switch using Through-silicon vias (TSVs). During read operations, the switch is configured to transfer DNN weights, input data, or intermediate results from MBs into LBs. During write operations, the partial results computed by each LB can be aggregated in the switch using combine blocks (Figure 9) before being stored in MBs. As we will show later in the paper, this on-switch combining capability greatly reduces MB access requirements. Additionally, in

Section IV-B, we show how the programmable nature of the switch allows for great flexibility in the types of computation (*e.g.,* DNN layer types) that can be implemented. In Section V, we compare using many small SAs in Maestro against a single large SA for two real-time inference application scenarios where the input batch size is 1.

The main contributions of this paper are:

- Formulating the LB-MB communication requirement in using many SAs for heterogeneous workloads (Section III).
- The Maestro memory-on-logic 3D-IC architecture to address this LB-MB communication requirement based on three innovations: switched memory-on-logic architecture (Section III-A), using H-trees to implement the switch (Section III-B), and on-switch elementwise combining (Section III-D).
- Implementation of a logically 3D H-tree switch using a regular 2D layout (Figure 6).
- A logic and layout design of a baseline Maestro system for performance assessment (Section V-D) and energy efficiency analysis (Section V-G).
- The "tile and pipe" computation paradigm (Figure 2) in scheduling SAs for tiled matrix computations and the associated scheduling algorithm (Section IV-A).
- Use examples for ShiftNet and Transformer models (Section IV-B) and results in substantially reduced latency (16× and 12×, respectively) when compared against a single large SA (Section V).

II. Background and Related Work

In this section, we first describe a tile and pipe computation paradigm which Maestro aims to support. Then, we discuss related work on 3D-IC architectures for DNNs. Finally, we provide background on ShiftNet [17] for computer vision tasks in Section II-C and the Transformer [16] for natural language processing (NLP) tasks in Section II-D.

A. Title and Pipe Paradigm

We consider matrix multiplication, which represents the bulk of DNN inference computation (see, *e.g.,* TPU [6]). To perform matrix multiplication on large weight and data matrices using smaller fixed-size systolic arrays, the matrices must be tiled as shown in Figure 2. Matrix multiplication can then be performed in three steps. First, tiles of the weight matrix (*e.g.,* 1, 2, 3, and 4) are loaded into the SAs. Then, tiles of the input/intermediate data matrix (*e.g.,* a and b) are piped into the SAs to perform matrix multiplication with the preloaded weight tiles. Each SA generates partial results which are added together in an elementwise fashion (*e.g.,* 1a + 2b) before the combined result is written to the MB. Elementwise combining is a distinguishing feature of our tile and pipe paradigm. This approach can be extended to support matrices of any size, as denoted by the dots in the figure. In Section IV-A, we describe how this tile and pipe paradigm is used to schedule the computation across all layers of a DNN on Maestro.

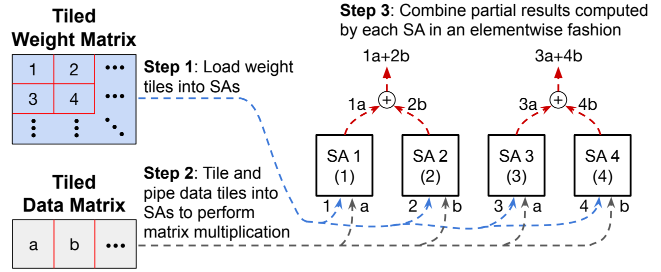


Fig. 2: The "tile and pipe" computation paradigm including an on-switch combining function.

B. 3D-IC Memory-on-Logic Architectures for DNN Inference

There are several prior projects which propose 3D-IC architectures for DNN inference. Like Maestro, these approaches use a memory-on-logic architecture for high bandwidth communication between one or more memory dies and the processing elements (PEs), which correspond to systolic cells in a systolic array, on a single logic die. Neurocube [7] uses multiple DRAM dies which are partitioned into a two-dimensional grid of memory vaults. Each memory vault can communicate locally with an associated group of PEs on the logic die which are arranged as an one-dimensional SA.

Tetris [4] uses the Neurocube architecture for DRAM memory dies, but arranges the PEs on the logic die as a two-dimensional grid for higher computational efficiency. Additionally, it introduces logic on the memory die to allow for summation between previous partial results stored in memory and new partial results from the logic die. In Maestro, we use a combine block (Figure 9) to sum partial results across multiple SAs before writing to memory. Unlike Tetris, which restricts the summation between a local pair of memory and logic, Maestro can perform summation across all SAs on the logic die, which is facilitated by on-switch elementwise combining (Section III-D), before writing the result to any MB.

C. ShiftNet for Computer Vision

ShiftNet [17] is a highly efficient Convolutional Neural Network (CNN), which is used in this paper as an evaluation case for Maestro. Figure 3a shows a single convolution layer trained with shift convolution. At the beginning of the layer, each channel in the data matrix is shifted a small amount based on a fixed offset. Matrix multiplication is then performed between the shifted data matrix and a 1×1 convolutional filter matrix. After convolution, batch normalization, and ReLU activation are applied. In Section V, we use ShiftNet to evaluate the performance gain of Maestro.

Due to the relatively large input size of samples in ImageNet [3] (3×224×224), the first convolution layer represents a significant portion (10-15%) of the total multiplier-accumulator (MAC) operations in a CNN such as ShiftNet. However, this layer is hard to implement efficiently on a single large SA, as there are only 3 input channels, meaning that most of the columns in an SA will be unoccupied. Recently, Xilinx proposed the use of two SAs for CNN inference to solve this

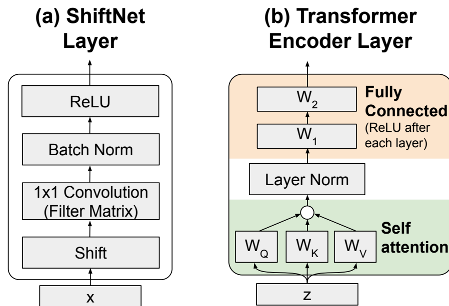


Fig. 3: ShiftNet convolution layers (a) and Transformer encoder layers (b) are evaluated in this work.

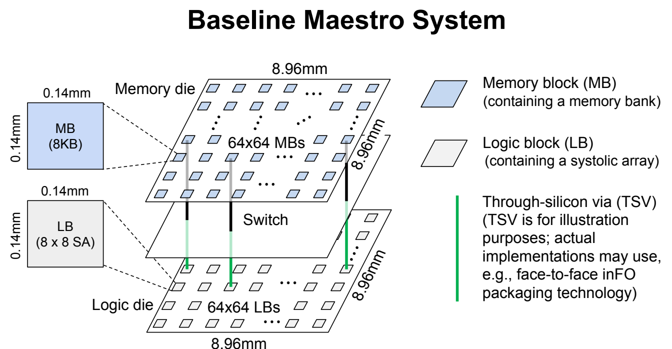


Fig. 4: The baseline Maestro system connects memory blocks (MBs) on the memory die to logic blocks (LBs) through a switch layer implemented with TSVs.

problem, with one SA specially designed for the first layer and the other SA used for the remaining layers [18]. Instead of using a specialized SA to handle this type of CNN layer, Maestro uses a collection of smaller SAs of the same size that can efficiently handle all layers in the CNN, while also being able to support other DNNs such as the Transformer discussed in Section II-D.

D. Transformer for Natural Language Processing

Our second evaluation case for Maestro in this paper is the Transformer [16]. Recent Transformer-based models have led to substantial accuracy improvements for NLP tasks over previous Recurrent Neural Network (RNN) models [14]. However, these Transformer models have significantly higher memory and computational cost than RNN models. For instance, GPT-2 [15] has 1.5 billion weights which is 16.1× larger than previous state-of-the-art RNN model (ELMo [14]). The Transformer for language translation tasks has an encoder-decoder structure. An encoder layer, shown in Figure 3b, is composed of a self-attention step, which weights relationships between word pairs in a sentence input, followed by Layer Normalization and two Fully-Connected layers. The self-attention step requires the same input to be multiplied by three relatively small learned matrices (W_Q, W_K, W_V). In Maestro, through the use of multiple smaller SAs, all three matrices in an encoder layer can be performed efficiently in parallel, which is not possible for a single large SA. In Section V, we show that Maestro can support low-latency inference for these large Transformers models by achieving high SA utilization on the small matrix multiplications.

III. Maestro 3D-IC Architecture

In this section, we describe the Maestro architecture and its subsystems in support of efficient and flexible LB-MB communication. In addition, we describe a baseline Maestro system which we conduct performance analysis on in Section V.

A. Maestro System Overview

The baseline Maestro system, shown in Figure 4, consists of a memory die with 64×64 SRAM Memory Blocks (MBs) stacked on top of a logic die with 64×64 Logic Blocks (LBs), which are interconnected through a switch using high-speed bit-serial Through-silicon vias (TSVs). For a given MB, its corresponding MB is the MB directly above the LB. This baseline is targeted for a 28 nm process node implementation running at 1 GHz and is used for sizing and performance assessment reported in this paper. Each LB contains one systolic array (SA). The systolic cells in each SA implement 8-bit fixed-point MAC using a bit-serial design [8], [11]. Throughout the paper we use TSVs to illustrate the use of 3D packaging. Other packaging technologies such as TSMC Integrated Fan-Out (InFO) may also be considered. Note that SRAM may be replaced with some other memory system such as MRAM (reduced cost and power, non-volatile, etc.).

A basic advantage of using 3D packaging technology is that LBs, MBs, and the switch do not have to all be on the same die. This avoids longer wiring in connected these elements; see arguments in [9].

For illustration simplicity, only 3 of the 4,096 full-duplex TSVs are shown in Figure 4 (their number are doubled for simplex TSVs). Each LB contains an 8×8 bit-serial SA [8] and uses a separate bit-serial TSV to connect to the switch. A TSV and MB can sustain memory access bandwidth requirements of 2 GB/s for an 8×8 SA with 8 bit-serial inputs and 8 bit-serial outputs running at 1 GHz. The remaining MB bandwidth (6 GB/s) is used for double buffering with external DRAM. That is, while performing the current computation, the Maestro system can output the result of the previous computation and input weights/data/programs for the next computation. Additionally, the remaining TSV bandwidth (1.52 GB/s) is used for loading weights as well as control for the next computation.

Figure 5 shows the Maestro switch in greater detail. The switch layer is shown as a 3D stack of H-trees (H-shaped trees). The memory blocks are interconnected by these H-trees placed on the memory die, which allows for coordinated high-speed communication between SAs and MBs (only two H-trees are shown). A switch point is placed at each joint of an H-tree, which can connect/disconnect its associated joint to control the data flow of the tree in support of local mode operations (Figure 8).

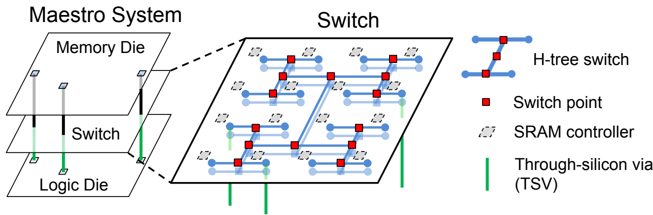


Fig. 5: The switch in Maestro is implemented using H-trees (one for each MB or a group of MBs).

We use H-trees, a popular layout structure previously used in distributing clock signal, because of their regular and scalable design. For example, we can embed multiple H-trees in a 2D space (Figure 6) and implement selection and combining functions (Figure 9) with a regular layout and efficiently implement the local mode (Figure 8). Other switch-efficient networks, such as Beneš networks [2], which have fewer switching points, are generally harder to lay out due to irregular wiring. Meshes and tori also support regular layouts, but do not provide connections for tree topologies to support low-hop routing.

B. 2D Implementation of Switch

For clarity of presentation, the two H-trees in Figure 5 are shown in a 3D perspective, with a dark blue H-tree on top of a light blue H-tree. However, it may be impractical to provide each H-tree with a physical die in a 3D-IC embodiment. Therefore, in practice, it could be desirable to lay out a number of these H-trees in 2D on a single die.

Figure 6 shows the layout process which enables multiple H-trees to achieve a regular 2D layout. Figure 6a depicts two H-trees in a 3D perspective as shown in Figure 5. In Figure 6b, these H-trees are placed on a 2D layout, by shifting the red H-tree down and right by a constant amount. This process can be repeated to support more H-trees as in Figure 6c with four H-trees. Finally, Figure 6d shows how multiple H-trees can be implemented in a regular fashion on a 2D space with two metal layers.

Figure 7a shows a read operation for Maestro, where data is read from SRAM into a systolic array. The memory controller fetches 8-bit data from SRAM into a bit-serial converter, which delivers the data to a demultiplexer in a bit-serial fashion. The data is then forwarded to the selected H-tree. During read operations, the combine block acts as a multiplexer, which selects one of the H-trees and forwards it to the systolic array through a TSV. The systolic array can then begin processing after receiving the input from the combine block.

Figure 7b shows a write operation for Maestro, where the result of a matrix multiplication performed on the systolic array is written back to the SRAM. The bit-serial outputs of the systolic array are forwarded to the H-tree chosen by the demultiplexer. The combine block is used to add partial results from the H-trees (discussed in Section III-D). The results from the combine block are then written into SRAM on the MB.

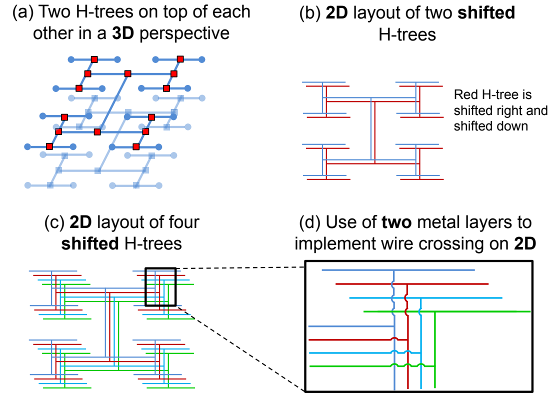


Fig. 6: Two H-trees shown in a 3D perspective in (a) are implemented in a 2D layout as shown in (b). (c) illustrates a case for four H-trees, where wire crossing is implemented with two metal layers as illustrated in (d).

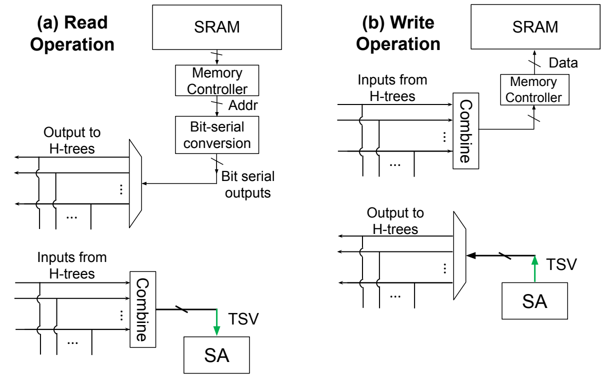


Fig. 7: (a) Memory read and (b) memory write operations. A detailed drawing of the combine block is shown in Figure 9.

C. Configuring Switch for Local Mode

Maestro can be configured to run in local mode by turning off middle connections on each H-tree. In Figure 8, a single H-tree on the memory die is shown overlaid on the logic die. The red squares denote connection points internal to the tree. In global mode (left), all leaves on the H-tree are connected. By turning off some of the connection points (the white squares in the middle of the figure), Maestro is able to run in a local mode (right). Under local mode, multiple groups of LBs and MBs may operate in parallel, each using their own sub H-trees. Computation within a group may have a reduced system latency, as data is required to traverse only its sub-trees rather than the entire H-trees.

D. On-switch Elementwise Combining

During a write operation (as shown in Figure 7b), the partial results from each LB, carried on multiple H-trees, can be combined in an elementwise fashion before being saved to the MBs. Figure 9 shows the design of the selection and combining circuitry on a 2D layout. The output from an LB (green line) is sent over a TSV to the red selection points

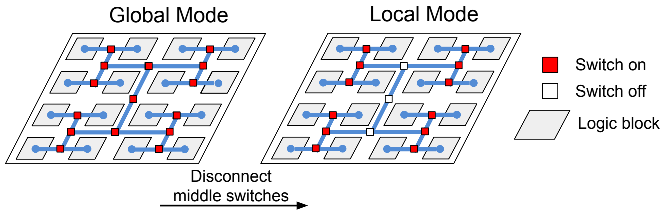


Fig. 8: Maestro supports parallel reads and writes between local groups of MBs and SAs by disconnecting middle switches on an H-tree.

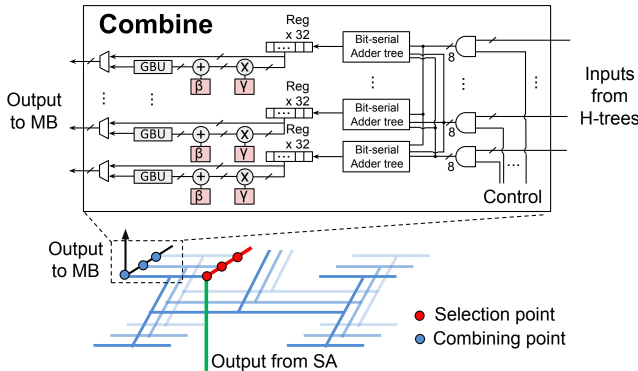


Fig. 9: Output from an SA is sent along the TSV (in green) to selection points (red circles). The combining points (blue circles) are implemented in a combine block to add partial results in an elementwise fashion.

(one per H-tree). Based on a predetermined routing schedule (discussed in Section IV-A), one of the H-trees is selected to transmit the partial results computed by the LB. This selection process is performed in parallel for the partial results computed by each LB.

The combine block takes input from the selected H-trees and performs an elementwise summation using the adder trees. The output can optionally be normalized with row-wise mean μ and row-wise standard deviation β stored in the combine block before being passed through a General-purpose Bit parallel Unit (GBU) for non-linear operations (e.g., ReLU, softmax). We could use Coarse Grain Reconfigurable Array (CGRA) for the GBU to allow fast reconfiguration. Finally, the output from the combine block is saved to the MB.

E. Multi-stage Combining

Note that a large number of H-trees will introduce a great amount of fan-ins to each combine block. For example, when there are 4,096 H-trees, the combine block will need 4,096 corresponding inputs, leading to a circuit design with excessive power and area. To mitigate this problem, we design a multi-stage combining operation shown in Figure 10 (two-stage combining is shown). The MBs are divided into multiple groups. For each group, one of the MBs is selected as the group leader (shown in red in Figure 10), which connects each of the other MBs in the same group via a set of H-trees. Each group leader can also access every MB on the memory die by using a separated set of H-trees. During the combining operation at a destination MB (shown in green in Figure 10), each group leader first accumulates the partial sum over its local MBs (stage 1), and then the partial sums from the group leaders are combined and delivered to the destination MB (stage 2).

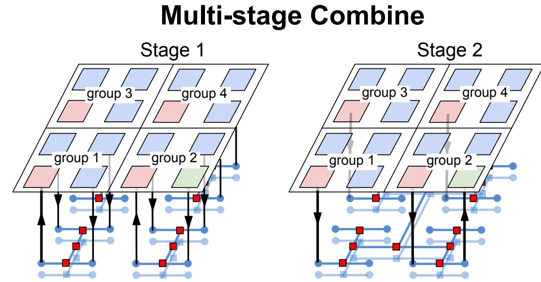


Fig. 10: Maestro supports multi-stage combining (a two-stage combining process is shown here).

Through two-stage combining, the number of fan-ins of a combine block are greatly reduced, since each MB only needs to connect with the MBs within the same group and all the group leaders, rather than all the MBs on the memory die. For a memory die with 64×64 MBs and 64 groups, two-stage combining decreases the number of fan-ins at each combine block from $4,096$ to $64 + 64 = 128$.

IV. Scheduling DNN Computation on Maestro

In this section, we discuss how DNN computation schedules are generated on Maestro and provide examples of computation being performed using multiple LBs and MBs.

A. Using Tile Dependencies for Schedule Generation

As shown in Figure 2, computation in Maestro operates at a tile level. Since the size of each weight matrix in a DNN and the size of input to the DNN is known ahead of time, a schedule can be precomputed which determines the LB on the Maestro system for each tile computation. This also requires knowledge of the Maestro configuration (*e.g.,* 4,096 8×8 SAs shown in Figure 4) in order to set the size and number of concurrent tiles being processed. Figure 11a shows how two weight and data matrices are tiled for computation on Maestro. In this example, each SA is 64×64. Therefore, the layer 1 weight matrix W_1 of size 128×128 is required to be partitioned into four tiles (1, 2, 3, and 4). At runtime, each tile will be loaded into an SA before being multiplied with a data tile. Correspondingly, the input data to the network (I) is tiled into two tiles (a and b). Layer 2 is tiled in a similar fashion to layer 1.

Figure 11b shows the tile dependency graph for these two weight and data matrices after the tiling procedure. Each vertex in the graph represents a tile and the directed edges show the dependencies between tiles. Each partial result tile (yellow) is generated by one data tile (grey) and one weight tile (blue). For instance, weight tile 1 and data tile a generate partial tile 1a.

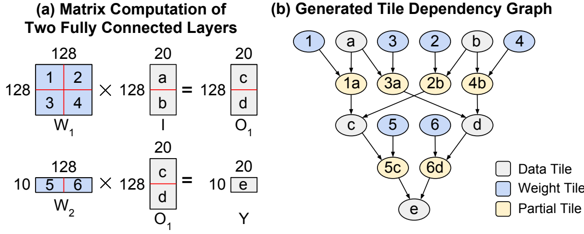


Fig. 11: (a) The matrix computation of two fully connected layers tiled for a systolic array of size 64×64. (b) The tile dependency graph for the two matrix multiplications.

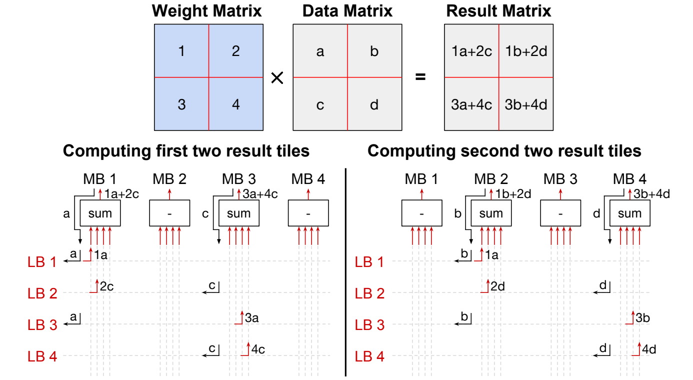


Fig. 12: Tiled matrix multiplication (2×2) in Maestro performed in two stages. The sum in a box denotes elementwise summation in a combine block before being written to an MB.

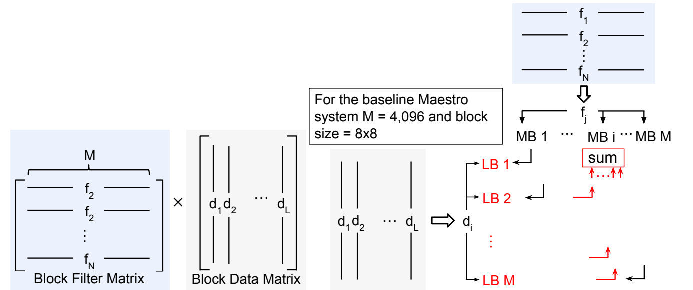


Fig. 13: Tiled matrix multiplication on Maestro supports arbitrary matrix shapes and sizes. Each element in the block filter matrix and block data matrix is a tile. As shown on the right, these blocks can be continually piped into Maestro.

Multiple partial tiles that are input edges into a data tile must be summed in an elementwise fashion in order to produce the tile. In the figure, 1a and 2b are added together to produce data tile c. This elementwise addition is scheduled to be performed on a specific combine block, as shown in Figure 9, before being written to the MB. Once data tile c is complete, the corresponding partial tile 5c, which depends on c, can be scheduled. Since execution of all tiles is deterministic, cycle accurate scheduling for the entire DNN can be performed in this manner. The scheduler may pipeline these elementwise combining operations over multiple graph nodes.

B. *Computation Patterns for DNN Inference*

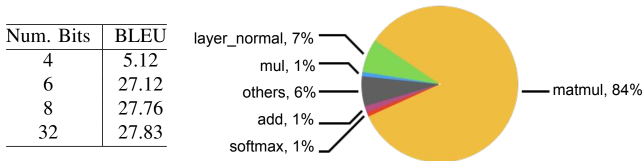
Now that we have described the tiling procedure for Maestro, we will show how matrix multiplication can be tiled in this manner and computed on the LBs. For simplicity, the examples in this section use a Maestro configuration with only 4 MBs and 4 LBs. In illustrations, the width and height of each SA is omitted.

Figure 12 demonstrates how Maestro is configured to perform tiled matrix multiplication for 2×2 tiles of the weight matrix in a fully connected layer, such as those in the Transformer network. In this example, the weight matrix is partitioned into four tiles (one vertical partition and one horizontal partition) denoted as 1, 2, 3, 4. The data matrix is partitioned in a similar fashion into tiles a, b, c, d. The weight matrix tiles are first preloaded into the LBs. Then, data tiles a, and c are loaded from MB 1 and MB 3, respectively, into the switch. LB 1 and LB 3 are configured to read data tile a, while LB 2 an LB 4 read data tile c. Matrix multiplication is then performed on these tiles, producing partial result tiles 1a, 2c, 3a, 4c. During the write, these four partial tiles are summed elementwise in the switch to produce two result tiles (1a+2c and 3a+4c) which are written to MB 1 and MB 3. This step is the repeated for the other two result tiles (1b+2d and 3b+4d).

Figure 13 shows how Figure 12 can be extended to support matrices of arbitrary sizes. In this figure, each element in the block filter and block data matrices represents a tile. Using this notation, the number of block matrix multiplications is MNL. With M LBs, Maestro performs all block computations in the minimum number of steps, i.e., NL, in two nested loops, where L is the height of the block data matrix and N is the width of the block filter matrix. In each of the L outer loops, there are N inner loops. In each inner loop, each LB loads input from an MB, computes a block matrix multiplication, and outputs the result to be combined for a destination MB, as shown on the right of the figure. As a convolution layer is represented as matrix multiplication when being processing with systolic arrays (see, *e.g.,* [6], [8], [11]), the tiling approach shown in Figure 12 and 13 naturally supports CNNs.

V. Evaluation

In this section, we first describe the experimental setup for the networks (ShiftNet and Transformer) used to evaluate Maestro. Then, we show the impact of quantization on the Transformer in terms of accuracy and provide a layerwise runtime breakdown for a GPU implementation. Next, we give an area and power breakdown for the baseline Maestro system described in Section III. Finally, simulation results for Maestro on quantized (8-bit fixed-point) ShiftNet and Transformer are compared to a single large SA in terms of latency, SA utilization, inference efficiency (GOPS/second/W), and energy efficiency (GOPS/W). For all evaluation results, we use a batch size of 1 to simulate an online scenario where the real-time nature of the application requires immediate feedback and samples cannot be buffered to form larger batches.



Num. Bits	BLEU
4	5.12
6	27.12
8	27.76
32	27.83

Fig. 14: (left) BLEU scores for different quantization bits (for both weights and activation) on the base Transformer model [16] for the English-to-German translation task (newstest2014). (right) Runtime breakdown for operation type in the Transformer running on one NVIDIA GTX 1080 Ti GPU.

Components	Location	Area (um^2)	Power (in percentage)
SRAM (8KB)	MB	10575	11.7%
SRAM controller	MB	2410	18.0%
Combine Block	MB	4037	30.2%
Switch Units	MB	1212	1.83%
TSV	MB, LB	44	N/A
SA	LB	14641	38.2%

TABLE I: Area and power breakdown for baseline Maestro system shown in Figure 4. For TSV, we do not provide a power estimates because design software and library do not currently report them.

A. *ShiftNet and Transformer Experimental Setup*

For ShiftNet [17] and Transformer [16], we use the baseline network settings presented in these two cited papers. For ShiftNet, this is a 24 layer network with 4.1 million learnable parameters (Table 6 in [17]). As discussed in Section II-C, ShiftNet replaces standard convolution with a shift operation followed by $1{\times}1$ convolution layers. We use our own PyTorch [13] implementation of ShiftNet (pytorch v1.0), which is available at https://github.com/BradMcDanel/column-combine. The input image size to ShiftNet is $3{\times}224{\times}224$.

For Transformer, we use the official TensorFlow [1] (v1.8) implementation.[1] The baseline Transformer has 6 encoder and 6 decoder layers with 65 million learnable parameters (Table 4 in [16]). For the English-to-German translation task, we use a 55 word input sentence and 100 word output sentence for Transformer.

B. *Impact of Quantization on Transformer*

Recently, it has been shown that CNN weights and activation quantization achieves large storage and computational savings over floating-point representations [5]. However, to the best of our knowledge, there was no quantized version of the Transformer. In this paper, we quantize both weights and activation values via uniform quantization. Given a tensor of weights $\mathbf{W}$, quantized $\hat{\mathbf{W}}$ is computed by:

$$\text{scale} = (\max(\mathbf{W}) - \min(\mathbf{W}))/(2^8 - 1)$$
$$\hat{\mathbf{W}} = \lfloor \frac{\mathbf{W} - \min(\mathbf{W})}{\text{scale}} \rceil \times \text{scale} + \min(\mathbf{W})$$

where $\lfloor\cdot\rceil$ rounds to the nearest integer. For activations, we use the same quantization scheme but fix $\min(\mathbf{W})$ and $\max(\mathbf{W})$ to -2 and $+2$. We evaluate our quantization scheme on the base Transformer [16], using the popular performance metric BLEU (bilingual evaluation understudy) [12]. As depicted in Table 14a, our results show that 8-bit fixed-point weight and data quantization introduces a negligible performance loss for the Transformer of only $\sim$0.1. The same quantization scheme is used for ShiftNet, which follows linear quantization proposed in [10]. Using 8-bit fixed-point quantization leads to minimal degradation in classification accuracy (less than 0.5%) for ShiftNet.

[1] https://github.com/tensorflow/models/tree/master/official/transformer

C. *Profiling Transformer GPU Implementation*

Figure 14b shows runtime profiling results for an English-to-German translation task using an official TensorFlow implementation of the Transformer. On the GPU, the average translation time for a sentence is 0.945 seconds. The CUDA execution times for each type of operation is summed over the entire inference process in order to calculate the percentage contribution of each operation. For all GPU profiling results, a floating-point implementation is used. Additionally, this runtime profile does not include CPU operations, as they represent an insignificant portion of inference runtime. Figure 14b illustrates the importance of speeding up matrix multiplication (matmul) as it represents 84% of the total runtime. As described earlier, Maestro performs matrix computations of each attenion layer efficiently by using many smaller SAs. Additional targets for speedup include layer normalization (layer_normal), element-wise multiplication (mul), and element-wise addition (add). Maestro can also implement these computations efficiently using its combine blocks as described in Section III-D.

D. *Area and Power Analysis*

We have designed the logic and layout for Maestro using the Synopsys Design Compiler with TSMC 28nm Library and CACTI-P. We use CACTI-P to simulate the SRAM and Synopsys Design Compiler to synthesize the other components including the systolic array and combine block. Table I summarizes the area and power breakdown for major components of each MB and LB. A significant fraction of MB area is consumed by the SRAM, which takes 54%, followed by combine block (20.6%), SRAM controller (12.3%) and TSV group (0.22%). The logic block area is mostly consumed by the systolic array. In terms of power, the SA and combine block contribute to most of the power consumption (38.2% and 30.2%), followed by the SRAM controller (18.0%), SRAM (11.7%), and switch units (1.83%).

E. *Impact of Data Tile Size for ShiftNet*

For large image datasets, such as ImageNet with a commonly used image resolution of $3{\times}224{\times}224$, the data matrix has significantly more rows ($224{\times}224$) than columns (9 in the case of $3{\times}3$ convolution). Due to this, the utilization of a large SA will be poor for this layer, as most of the columns

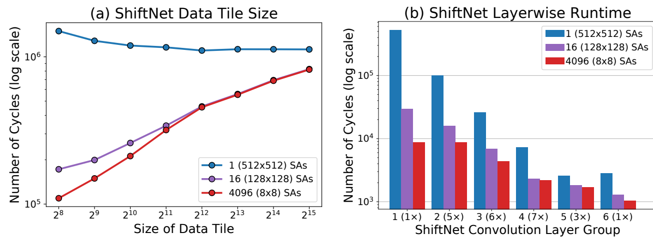


Fig. 15: (a) Smaller data tiles through vertical data tiling substantially improves the runtime ShiftNet. (b) Most of the runtime is spent processing the first layer. See Table VI in [17] for ShiftNet architecture details. 2 (5×) means that this layer group 2 is a layer repeated 5 times.

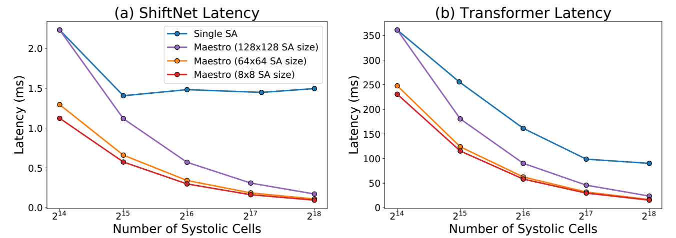


Fig. 16: Latency in milliseconds for processing one sample with a single SA and Maestro configurations at the same number of systolic cells for ShiftNet (a) and Transformer (b).

in the SA will not be used. However, for multiple smaller SAs, the tall data matrix can be partitioned into smaller tiles, which decreases the runtime processing the layer. For instance, using 16 SAs, the data matrix could be partitioned into 16 tiles (each 3×56×56), which reduces the runtime of that layer by a factor of 16. Figure 15a shows the importance of vertical data tiling on reducing the runtime of ShiftNet inference when used in Maestro. The blue line shows a setting for a single SA of size 512×512. The purple and red lines represent two Maestro configurations, with 16 (128×128) SAs and 4096 (8×8) SAs, respectively. All three settings have the same total number of systolic cells (262,144).

For the two Maestro settings, as the size of the data tile decreases on the x-axis, the total number of cycles required to perform inference for one sample decreases. However, the single SA setting actually has an increased runtime as the data tile size is reduced. Since the single SA can only process one tile at a time (regardless of its size), data tiling provides no benefit. Instead, the single SA must pay additional runtime due to data skew inherent when processing with systolic arrays, increasing the runtime. This illustrates one of the main benefits of Maestro: by using multiple small SAs, the system can adjust to better fit the matrix computation through fine-grained tile operations.

Figure 15b shows a layerwise runtime breakdown for the same three settings in Figure 15a. In this experiment, a data tile size of 2^{15} is used for the single SA setting and a data tile size of 2^{8} is used for the 16 (128×128) and 4096 (8×8) settings. The majority of inference runtime is spent in the first several layers. Since the weight matrices in these layers are smaller, a single large SA cannot by fully utilized, leading to a longer runtime. Through data tiling, Maestro is able to reduce the runtime of these layers, by processing portions of the input to these layers with multiple small SAs in parallel.

F. Maestro Latency Reduction

We compare the latency achieved by the many smaller SAs in Maestro to a single large SA on both ShiftNet (Figure 16a) and Transformer (Figure 16b). For a fair comparison, on any given point on the x-axis, all settings use the same number of systolic cells. The majority of the latency reduction in Figure 16a is explained by data tiling as discussed in Section V-E. As the number of SAs is increased (*e.g.,* from 128×128 to 8×8) additional reduction in latency is achieved through more fine-grained tiling.

For the transformer latency results in Figure 16b, the smaller SAs are better utilized for many of the smaller matrices in the Transformer, such as the W_{Q}, W_{K}, and W_{V} matrices (each of size 64×512) in the self-attention step. Additionally, the computation for these matrices in a single encoder layer can be performed in parallel, which is not possible for the single large SA. Finally, for these smaller matrices, the single large systolic array is often underutilized.

Figure 17 shows the average utilization over all SAs for the same configurations in Figure 16. We can see that the reduction in latency achieved by the Maestro settings in Figure 16 is due to maintaining higher SA utilization as the number of cells is increased (Figure 17). We note from Figure 16 at the number of systolic cells equal to 2^{18}, the baseline Maestro configuration achieves impressive latency reduction for ShiftNet and Transformer at 16× and 12×, respectively. The latency for Transformer inference is ∼20 ms, as opposed to 0.945 seconds noted earlier for a floating-point GPU implementation (a 47× improvement). Similarly, the ShiftNet latency for the baseline Maestro configuration is 0.09 ms versus 2.7 ms for the floating-point GPU implementation (a 30× improvement). Note that since we consider an real-time scenario (batch size of 1), throughput is simply 1 / latency.

G. Energy Efficiency of Maestro

We compare the energy efficiency of the baseline Maestro system (4,096 8×8 SAs) in Figure 4 against a single large SA of size 512×512, as we did for latency in Section V-F. We note that in a parallel processing system when the number of fixed-size SAs scales up, unlike a 3D implementation where SAs can connect to their corresponding MBs in constant distance along the third dimension, a 2D implementation will suffer from long wires [9]. Thus, as the number of SAs increases, power and delay of a 2D implementation, due to increased wire lengths, will eventually dominate those of a 3D implementation. We argue below that even under a 2D implementation Maestro will be competitive in energy efficiency.

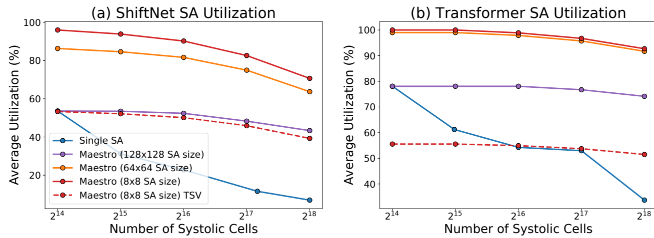


Fig. 17: SA utilization for ShiftNet (a) and Transformer (b). The dotted red curve is the TSV utilization for the 8×8 SAs shown on the solid red curve.

Assume a layout scheme resembling to that depicted in Figure 1 where LBs, MBs, and the switch are all on the same 2D plane and a die of twice the size is used to accommodate LBs as well as wiring and MBs. We simulate power consumption with Synopsys Design Compiler and CACTI-P.

The power for the baseline Maestro system is 1.36× higher than a single large SA with the same number of systolic cells (4,096×8×8). The power savings for the single large SA over Maestro are due to omitting the H-trees and the combine blocks. However, the decreased runtime per sample for Maestro (16× for ShiftNet and 12× for Transformer) translates to an 11.76× and 8.83× improvement in inference efficiency (samples/second/W) for ShiftNet and Transformer, respectively, over the single large SA. While Maestro consumes more power, the improved computational efficiency due to high-utilization use of many small SAs as well as on-switch elementwise combining leads to higher throughput and therefore higher inference efficiency.

Additionally, in terms of energy efficiency, the baseline Maestro system achieves 664.60 GOPS/W. This performance is due to the regular structure of H-tree and systolic arrays, which significantly reduces the propagation delay of critical paths and raises the throughput. In general, it is important to evaluate both energy and inference efficiency to measure the performance of a system. For deep learning applications, it is possible to do a large amount of work per sample in a way that achieves high GOPS/W (energy efficiency) but low samples/second/W (inference efficiency).

VI. Conclusion

Use of many small systolic arrays in parallel, as opposed to a single large one, can achieve high processor array utilization for heterogeneous workloads of varying shapes and sizes, such as those present in the ShiftNet and Transformer models. By leveraging short-distance vertical 3D-IC interconnects in the third dimension, the Maestro architecture proposed in this paper allows these systolic arrays to have high bandwidth, yet flexible, parallel access to multiple memory banks.

For Transformer, we have demonstrated that the baseline Maestro architecture can lead to an order of magnitude improvement (*i.e.,* 12×) in inference latency for natural language processing. For ShiftNet, Maestro achieves similar performance gains, i.e., 16× latency reduction. Our analysis shows that most of these gains are due to high processor array utilization resulting from the use of small systolic arrays enabled by the Maestro memory-on-logic 3D-IC architecture.

Maestro is novel in its switched memory-on-logic organization, H-tree based switch and on-switch elementwise combining functionality. Resulting from these features, the system can scale up the computation throughput for matrix computations by extending memory and logic dies along the horizontal dimension. This scalability is important in the post-Moores Law era, where we can only increase computation bandwidth by using increased chip area rather than reduced device size.

VII. Acknowledgments

This work is supported in part by the Air Force Research Laboratory under agreement number FA8750-18-1-0112, a gift from MediaTek USA and a Joint Development Project with TSMC.

References

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283, 2016.

[2] C. Clos. A study of non-blocking switching networks. *Bell System Technical Journal*, 32(2):406–424, 1953.

[3] J. Deng, W. Dong, R. Socher, L.-J. Li, et al. Imagenet: A large-scale hierarchical image database. In *CVPR '09*.

[4] M. Gao, J. Pu, X. Yang, M. Horowitz, et al. Tetris: Scalable and efficient neural network acceleration with 3d memory. In *ASPLOS '17*.

[5] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan. Deep learning with limited numerical precision. In *ICML '15*.

[6] N. P. Jouppi, C. Young, N. Patil, D. Patterson, et al. In-datacenter performance analysis of a tensor processing unit. In *ISCA '17*.

[7] D. Kim, J. Kung, S. Chai, S. Yalamanchili, and S. Mukhopadhyay. Neurocube: A programmable digital neuromorphic architecture with high-density 3d memory. In *ISCA '16*.

[8] H. T. Kung, B. McDanel, and S. Q. Zhang. Packing sparse convolutional neural networks for efficient systolic array implementations: Column combining under joint optimization. *ASPLOS '19*.

[9] H. T. Kung, B. McDanel, S. Q. Zhang, et al. Systolic building block for logic-on-logic 3d-ic implementations of convolutional neural networks. *ISCAS '19*.

[10] D. Lin, S. Talathi, and S. Annapureddy. Fixed point quantization of deep convolutional networks. In *International Conference on Machine Learning*, pages 2849–2858, 2016.

[11] B. McDanel, S. Q. Zhang, H. T. Kung, and X. Dong. Full-stack optimization for accelerating cnns using powers-of-two weights with fpga validation. *ICS '19*.

[12] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu. Bleu: a method for automatic evaluation of machine translation. In *ACL '02*.

[13] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

[14] M. E. Peters, M. Neumann, M. Iyyer, et al. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*, 2018.

[15] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language models are unsupervised multitask learners, 2019.

[16] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

[17] B. Wu, A. Wan, X. Yue, P. Jin, et al. Shift: A zero flop, zero parameter alternative to spatial convolutions. *CVPR '18*.

[18] Xilinx. Accelerating dnns with xilinx alveo accelerator cards. Technical report, Xilinx, October 2018.