# SemiMap: A Semi-folded Convolution Mapping for Speed-Overhead Balance on Crossbars

Lei Deng, *Member*, *IEEE*, Ling Liang, Guanrui Wang, Liang Chang, Xing Hu, Xin Ma, Liu Liu, Jing Pei, Guoqi Li, *Member*, *IEEE*, and Yuan Xie, *Fellow*, *IEEE*

*Abstract*—**Crossbar architecture has been widely used in neural network accelerators, involving conventional and emerging devices. It performs well on the fully-connected layer through efficient vector-matrix multiplication. Whereas, the advantages degrade on the convolutional layer with huge data reuse, since the execution speed and resource overhead are imbalanced when using existing fully-unfolded or fully-folded mapping strategy. To address this issue, we propose a novel semi-folded mapping (SemiMap) framework for implementing the convolution on crossbars. It simultaneously folds the physical resources along the row dimension of feature maps and unfolds them along the column dimension. The former reduces the resource overhead, and the latter maintains the parallelism. A feature map slicing scheme is further proposed to enable the processing of large-size image. Via our mapping framework, a row-by-row streaming pipeline for intra-image dataflow and periodical pipeline for inter-image dataflow are easy to be obtained. To validate the idea, we build a many-crossbar architecture with several designs to guarantee the overall functionality and performance. Based on the measurement data of a fabricated chip, a mapping compiler and a cycle-accurate simulator are developed for the hardware simulation of large-scale networks. We evaluate the proposed SemiMap on various convolutional neural networks across different network scale. >35x resource saving and several hundred times cycle reduction are demonstrated compared to the existing fully-unfolded and fully-folded strategies, respectively. This work jumps out of the current extreme mapping schemes, and provides a balanced solution on how to efficiently deploy the computational graphs with data reuse on many-crossbar architecture.**

*Keywords:* Convolutional Neural Networks, Deep Learning Accelerator, Crossbar Architecture, Network Mapping

## I. Introduction

The recent crossbar architecture targets integrating the compute and memory for efficiently executing the vector-matrix multiplication (VMM), which is the major operation in various neural networks (NNs). On crossbars, the VMM can be completed with shorter time (e.g. conventional memory array based virtual crossbar for near-memory computing [1]–[3]) or even only one cycle (e.g. emerging physical crossbar for in-memory computing [4]–[9]). In this way, it presents great potential for building high-performance system. A many-crossbar architecture is usually characterized as two design levels: (1) functional crossbar that is a self-contained small neural network mainly based on a memory crossbar for the weighted connections and its peripheral processing circuits;
(2) many-crossbar network wired by a scalable routing infrastructure. This architecture has been widely used in NN accelerators. In particular, most neuromorphic platforms [10], [11], including early analog/digital mixed circuits [12]–[14] or recent fully digital circuits [1]–[3], [15], [16], use this many-crossbar architecture with various routing topologies [1], [15], [17], [18]. Besides conventional technologies, plenty of researches leverage modified SRAM [8]/ Flash [9] or various emerging non-volatile memory devices with in-memory computing capability to design this many-crossbar architecture, such as the most widely used RRAM (resistive RAM) [4]–[6], [19]–[24], PCRAM (phase-change RAM) [7], [25], and MRAM (magnetic RAM) [26].

This crossbar based architecture naturally performs well on fully connected (FC) layers with dense VMM, i.e. fast throughput and high utilization. However, these advantages degrade on convolutional (Conv) layers since the operands are greatly reused. Usually, there exist two strategies for mapping the Conv layer onto the many-crossbar architecture: fully-unfolded and fully-folded. The former first converts the Conv layer to an FC layer and then assigns independent physical cells for all logical neurons and weights. This way is able to achieve very high throughput, but at the cost of extremely huge resource overhead. For example, for convolutional neural networks (CNNs) on medium-size CIFAR10 dataset [27], it uses more than 30,000 crossbars [28]; and for larger models [29]–[31] on ImageNet dataset [32], more than hundreds of thousands of crossbars [33] are usually required. On the other hand, the fully-folded strategy sufficiently leverages the data reuse of Conv layer. Specifically, it fully reuses the crossbar cycle by cycle for completing the Conv operations across many sliding windows [6]. This results in much less resource overhead but at the cost of very high latency. Taking a Conv layer with 224 feature map (FM) size and $3\times3$ weight kernel as an example, more than $5\times10^4$ cycles are needed. In short, the above two mainstream mapping strategies are difficult to achieve a satisfactory balance between the execution speed

The work was partially supported by National Natural Science Foundation of China (Grant No. 61475080, 61603209, and 61876215), National Science Foundation (Grant No. 1719160, 1725447, 1730309, and CCF 1740352), and SRC nCORE NC-2766-A. Lei Deng and Ling Liang contributed equally to this work, corresponding authors: Guoqi Li and Yuan Xie. Lei Deng, Ling Liang, Xing Hu, Xin Ma, Liu Liu, and Yuan Xie are with the Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA 93106, USA (email: leideng@ucsb.edu, lingliang@ucsb.edu, xinghu@ucsb.edu, xinma@ucsb.edu, liu_liu@ucsb.edu, and yuanxie@ucsb.edu). Guanrui Wang, Jing Pei, and Guoqi Li are with the Center for Brain Inspired Computing Research, Beijing Innovation Center for Future Chip, & Department of Precision Instrument, Tsinghua University, Beijing 100084, China (email: wgr16@mails.tsinghua.edu.cn, peij@mail.tsinghua.edu.cn, and liguoqi@mail.tsinghua.edu.cn). Liang Chang is with Fert Beijing Research Institute, BDBC, School of Electronic and Information Engineering, Beihang University, Beijing 100191, China (email: Liang.Chang@buaa.edu.cn).

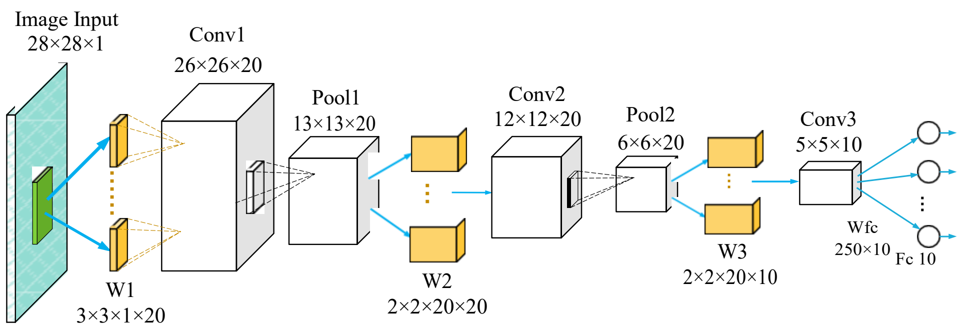


Fig. 1: Example of convolutional neural networks.

and resource overhead. However, this is highly required by some application scenarios. For instance, the performance of unmanned aerial vehicle heavily depends on the real-time vision processing capability under great resource and energy limitation.

To address the imbalance issue on current many-crossbar platforms is challenging. For the ones based on asynchronous communication [1], [15], the timing schedule is very simplex. Usually, the time step with identical operation flow is the minimum time unit. This makes it difficult to implement complex timing pattern for resource reuse. For other synchronous ones [6], the limited scalability makes it difficult to unfold the whole computational graph. To this end, we propose a semi-folded mapping (SemiMap) framework with slight modifications on current many-crossbar architectures. The 'semi-folded' represents that it simultaneously folds the physical resources along the row dimension of the FMs and unfolds them along the column dimension. The former is for resource saving, and the latter is to maintain parallelism. In particular, we observe that generating one output row in Conv layer only requires several input rows and a shared weight kernel. Hence, we propose to reuse the computation resources and weight memory along the FM row dimension to save resources significantly. Meanwhile, we duplicate the weight parameters and neuron computations along the column dimension which maintains the parallelism to a great extent. Moreover, a FM slicing scheme to process the large-size image is elaborated. The whole intra-image dataflow can be executed under a row-by-row streaming pipeline and the inter-image dataflow further forms a periodical pipeline. In this way, we achieve an ideal trade-off between the resource overhead and running speed. To guarantee the functionality and performance, a many-crossbar architecture with versatile vector and matrix operations, multi-phase-per-step timing schedule, integrating point-to-point (P2P) and adjacent multicast (AMC) routing strategies, and routing-aware neuron reservation, is designed. A real chip is fabricated to implement all these designs. Based on the measurement data, a mapping compiler and a cycle-accurate simulator are developed for the hardware simulation of large-scale networks. Various CNNs across different network scale are comprehensively evaluated. >35x resource saving and several hundred times cycle reduction are demonstrated compared to the existing fully-unfolded and fully-folded strategies, respectively. This work promises a general solution on how to efficiently deploy the computational graphs with data reuse on many-crossbar architecture.

The rest of this paper is organized as follows: Section II introduces some backgrounds of CNNs and crossbar architecture; Section III provides the basic operations and routing modes, introduces existing mapping schemes, and explains the proposed SemiMap framework in detail; The layer-level and network-level mapping results, routing optimization, and performance comparisons with state-of-the-art platforms, are reported in Section IV; Finally, Section V concludes and discusses the paper.

## II. Convolutional Neural Network and Crossbar Architecture

### A. Convolutional Neural Network

Compared to 1D multi-layered perceptron (MLP), CNN is specially designed for 2D data processing, such as image recognition. As shown in Figure 1, it usually includes three layer types: Conv layer, pooling layer (Pool), and FC layer. Conv layer generates the output FMs by executing the 2D sliding convolution operation, where each output FM is determined by all input FMs. All the sliding windows between one input FM and one output FM share the same weight kernel. Adjacent sliding windows are often overlapped according to the stride value. The number of output FMs are determined by the number of weight kernels, and the size of output FMs is co-determined by the size of input FM, weight kernel, padding and stride value. Pool layer is used for down-sampling the size of FMs (also introducing translation invariance), but the output FM is only determined by its corresponding input FM and the adjacent pooling windows are usually not overlapped, which are different from Conv layer. The FC layer is similar to that in MLP with dense VMM operation. The whole CNN computation can be described as

$$
\begin{cases}
Conv: \mathbf{FM}_n^{out} = \phi(b_n + \sum_m \mathbf{FM}_m^{in} \sim \mathbf{W}(m,n)) \\
Pool: \mathbf{FM}_n^{out} = Pooling(\mathbf{FM}_n^{in}) \\
FC: \mathbf{Y} = \phi(\mathbf{b} + \mathbf{XW})
\end{cases}
\quad (1)
$$

where, for Conv and Pool layers, $\mathbf{FM}_n^{out}$ is the $n$-th output FM, i.e. a 2D matrix of neuronal activations, $b_n$ is a bias item shared by all the neurons in $\mathbf{FM}_n^{out}$, $\mathbf{FM}_m^{in}$ is the $m$-th input FM, $\mathbf{W}(m,n)$ is a 2D weight kernel (e.g. $3 \times 3$) connecting $\mathbf{FM}_m^{in}$ and $\mathbf{FM}_n^{out}$, $\sim$ is a 2D convolutional operation, $\phi(\cdot)$ is

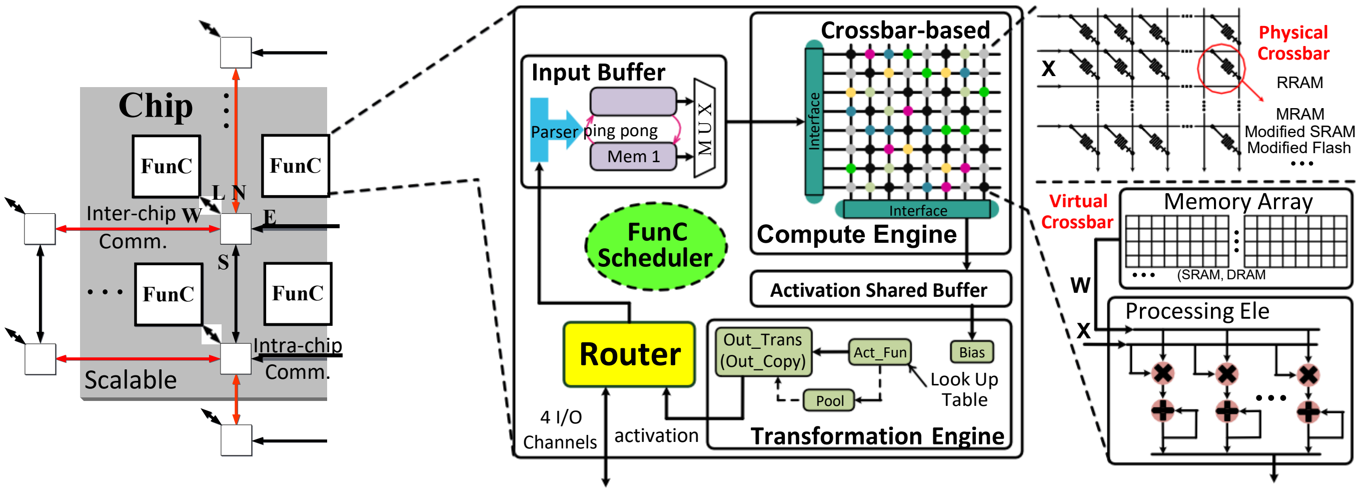


Fig. 2: Illustration of the crossbar-based architecture, wherein the crossbar can be either physical crossbar (e.g. RRAM [5], PCRAM [7], MRAM [26], as well as modified SRAM [8] or Flash [9], etc.) for in-memory computing or virtual crossbar built by conventional memory array (e.g. SRAM [1], [3] or DRAM [2]) and PEs for near-memory computing.

a nonlinear activation function, e.g. ReLU function $\phi(x) = max(x, 0)$; for FC layer, $\mathbf{X}$ and $\mathbf{Y}$ are the input and output vector of neuronal activation, respectively, $\mathbf{W}$ is the weight matrix, $\mathbf{b}$ is a bias vector, and $\phi(\cdot)$ is the same as that in Conv layer.

## B. Crossbar Architecture

The crossbar-based architecture is usually a hierarchical and scalable system. As shown in Figure 2, many independent functional crossbars (FunCs) are connected by the routing infrastructure to form a many-crossbar network.

*Functional Crossbar*. FunC is a basic build block of our crossbar-based architecture. It is a self-contained NN with a memory crossbar for weighted connections and its peripheral processing circuits, such as input buffer, compute and transformation engine, router, and scheduler. The input buffer provides the input activations for the compute engine and also buffers the output activations from the router (generated by local or remote transformation engines). There are two memory chunks here, assuming $N$ cells for each, acting as two ping-pong buffers switching between modes of router write and compute engine read. In this way, the computation and routing within one phase is decoupled for better parallelism. The compute engine occupies the major computation, i.e. multiplications and accumulations (MACs). It multiplies the input activation vector read from the input buffer and the weight matrix stored in the memory crossbar ($N \times N$ ) for completing the VMM operation. If the memory crossbar is physically built by aforementioned in-memory computing devices [5], [7]–[9], [26], the VMM is processed on the crossbar itself; while if conventional memory array (e.g. SRAM [1], [3]/ DRAM [2]) is used, extra processing elements (PEs, e.g. multipliers and accumulators) are required to execute the VMM operation, and the 'memory array & PEs' forms a virtual crossbar. A shared memory is used to buffer the calculated intermediate activation from the compute engine. Then, the transformation engine conducts the activation transformation, including adding bias, nonlinear activation function (realized by extra logics or look up table), possible pooling operation, and generates the output activation and sends it to the router. Router is a communication interface that connects four adjacent FunCs in a 2D-mesh network and the local FunC. Scheduler manages the whole timing sequence, wherein the compute and transformation engine can be enabled or disabled at each phase. The whole sequence is executed phase by phase with static weight matrix and dynamic activation dataflow.

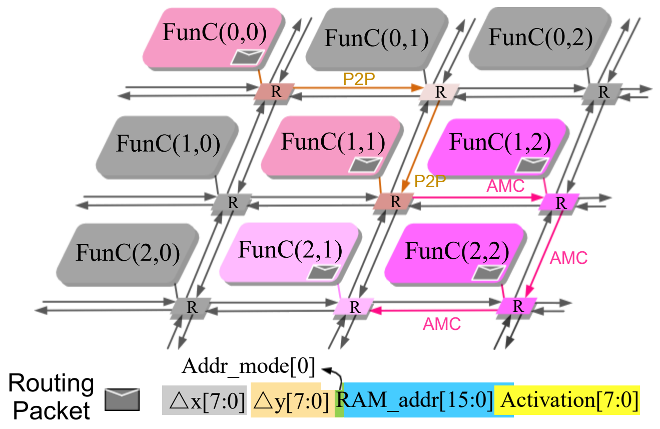


Fig. 3: Two routing modes: point to point (P2P) and adjacent multicast (AMC).

*Many-Crossbar Network*. FunCs communicate with each other via configuring the routing tables in every FunC to generate a netlist of neuronal connection topology. The target neurons can be in the local FunC or other intra-/inter-chip FunCs. As shown in Figure 3, the routing packet consists three segments: relative XY FunC addresses ($\Delta x$-8b; $\Delta y$-8b), address mode (1b) and RAM address (16b), and activation data (8b). Here the address mode means where to put the data in the target FunC (0-input buffer; 1-memory crossbar). When address mode is 0, the lower 8 bits in RAM address represent the row index in the input buffer, and the higher 8 bits are ignored; while when it is 1, the higher and lower 8 bits record the row and column index of the crossbar, respectively. On the 2D-mesh FunC plane, we use the coordinate format of $(y, x)$ to represent the FunC location, in which $y$ and $x$ denote the vertical and horizontal position, respectively. Dimension-ordered point to point (P2P) routing is the most widely used routing strategy, and the horizontal ($\Delta x$) direction has higher priority than the vertical ($\Delta y$) direction. For instance, FunC$(0, 0)$ sends a routing packet to FunC$(1, 1)$ whose P2P target address (relative) is directly programmed into the routing table before the FunC execution. In this case, $\Delta x = 1$ and $\Delta y = 1$ are initialized. The P2P routing parses $\Delta x$ firstly, thus sends the packet to FunC $(0, 1)$, and then parses $\Delta y$ and sends it to the destination of FunC $(1, 1)$. Every time passing an intermediate FunC router, the $\Delta x$ and $\Delta y$ will be updated until the packet reaches the destination FunC (i.e. $\Delta x = 0$ & $\Delta y = 0$). The two address modes and another routing mode (adjacent multicast routing) will be explained in Section III-A for more details.

## III. Semi-folded Mapping Framework

Before introducing the mapping framework, we design several operation modes and a multicast routing mode on the above many-crossbar architecture. Then we briefly recall the existing fully-unfolded and fully-folded mapping schemes. At last, in order to obtain a satisfactory balance between the speed and overhead, we propose the SemiMap framework in this section.

### A. Operation and Routing Mode

For supporting CNNs well, we design three vector and matrix operations in the compute engine and five transformation operations in the transformation engine, as listed in Table I.

TABLE I: Computation and transformation operations.

| Block | Operation | Definition |
|---|---|---|
| **Compute Engine** | VMM | $\mathbf{y} = \mathbf{W} \cdot \mathbf{x}$ |
| | VVA | $\mathbf{y} = \sum_i \mathbf{x}_i, i = 0, 1, ..., 127$ |
| | VB | $\mathbf{y} = \mathbf{x}$ |
| **Trans. Engine** | Bias | $\mathbf{y} = \mathbf{x} + \mathbf{x}_b$ |
| | Act_Fun | $\mathbf{y} = \phi(\mathbf{x})$ |
| | Pool | $y_i = \max/\text{ave}(\{x_j \mid j \in \text{pool}_i\})$ |
| | Out_Copy | $\text{Out\_data}_i = \text{Out\_data}_j$ |
| | Out_Trans | Send output to Router |

In the compute engine, besides the mentioned VMM operation, two other operations are also integrated. The vector-vector addition (VVA) operation reduces multiple vectors to one vector, that is useful for hierarchically accumulating large amount of FMs. In this case, one crossbar cannot complete the whole computation for generating one output neuron due to the limited crossbar size, then VVA can be used to accumulate the partial activations from multiple pre-VMM crossbars. Note that in VVA operation, the crossbar no longer stores the static weight matrix, instead, it stores the dynamic partial activations. The whole $N \times N$ crossbar splits to two $\frac{N}{2} \times N$ chunks for ping-pong buffer like the input buffer. In this sense, the routing data might have two destinations (the input buffer or the memory crossbar) that requires two address modes as mentioned in Section II-B. Different from VMM and VVA operations, the vector buffer (VB) operation totally bypasses the crossbar and just copies the activations from the input buffer to the transformation engine, which is used in Pool layer or the one just for neuron copy. In the transformation engine, Bias (adding bias), Act_Fun (activation function), Out_Copy (output copy), Out_Trans (output transmission), and Pool operations are designed to support all the mentioned activation transformations in CNNs. Here the Out_Copy is for the duplication of neuron outputs within the convolution overlap between adjacent column-wise slices (to be introduced in Section III-D).

```
Data: Received routing packet from local or adjacent
      FunC.
Result: Update and send it out, or put it into local FunC.
while Packet queue is non-empty do
    Read a packet;
    // P2P routing, routing priority $\Delta x > \Delta y$;
    if $\Delta x$ $(\Delta y)$     0 then
        if $\Delta x$ $(\Delta y) > 0$ then
            $\Delta x$ $(\Delta y) \Leftarrow \Delta x$ $(\Delta y) - 1$;
            Send it out to the Eastern (Southern) FunC;
            Continue;
        end
        if $\Delta x$ $(\Delta y) < 0$ then
            $\Delta x$ $(\Delta y) \Leftarrow \Delta x$ $(\Delta y) + 1$;
            Send it out to the Western (Northern) FunC;
            Continue;
        end
    else
        // AMC routing;
        if (AMC_$\Delta y$, AMC_$\Delta x$)     (0, 0) then
            Generate a new routing packet with $(\Delta y, \Delta x)$
              = (AMC_$\Delta y$, AMC_$\Delta x$);
            Send it out according to above P2P strategy;
        end

        if Addr mode=0 then
            Allocate the RAM_addr[7:0]-th cell in the
              input buffer;
        else
            Allocate the (RAM_addr[15:8],
              RAM_addr[7:0])-th cell in the crossbar;
        end
        Put the Activation[7:0] into the allocated cell;
    end
end
```

**Algorithm 1:** Routing Algorithm

On the routing side, due to memory limitation, the routing table cannot be very large. Only one target address is allowed for each neuron, which indicates it can only be connected to $N$ neurons in the target FunC at most. Different from the fan-in limitation that can be addressed by hierarchical accumulation through VVA operation, the fan-out limitation becomes an intractable problem. Existing schemes usually

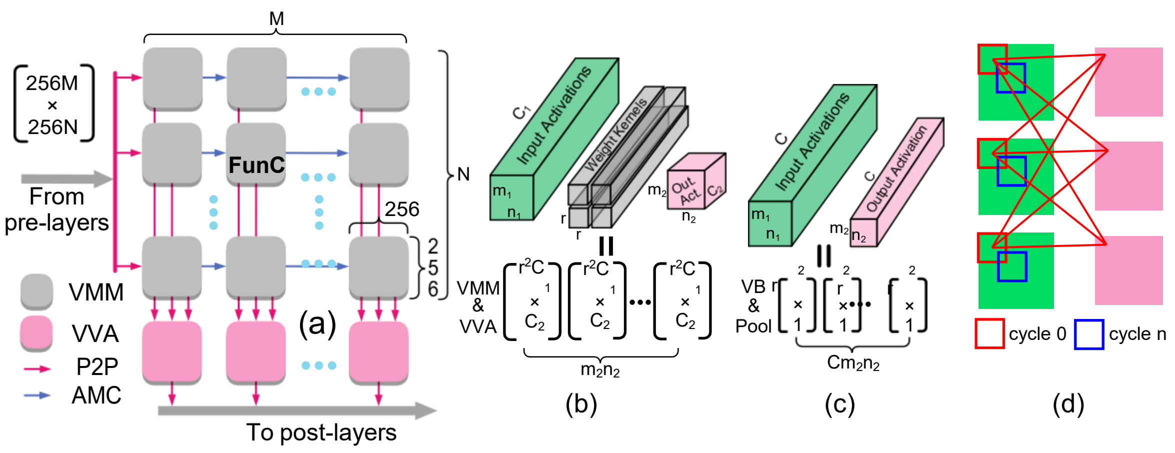


Fig. 4: Existing mapping schemes: (a) fully-unfolded mapping of FC layer; (b) fully-unfolded mapping of Conv layer; (c) fully-unfolded of Pool layer; (d) fully-folded mapping of Conv or Pool layer.

introduce copy neurons to solve it, i.e. Out_Copy operation which just copies the activation while disables any effective computation. This is useful for small amount of duplications, such as for the convolution overlap copy. However, this intra-FunC copy is inefficient for large amount of duplications. For instance in large models, each neuron is connected to thousands of post-neurons that require huge activation duplications. To this end, we additionally propose an adjacent multicast (AMC) routing mode. As shown in Figure 3, the packet propagation along FunC (1, 1), FunC (1, 2), FunC (2, 2), to FunC(2, 1) can be completed by this AMC routing. If we configure the AMC registers, i.e. (AMC_Δy, AMC_Δx), in FunC (1, 1), FunC (1, 2) and FunC (2, 2) to be (AMC_Δy=0, AMC_Δx=1), (AMC_Δy=1, AMC__Δx=0), and (AMC_Δy=0, AMC__Δx= 1), respectively, then FunC (1, 2), FunC (2, 2) and FunC (2, 1) are able to share the same packets received by FunC (1, 1) from FunC (0, 0). In this relay-like way, there is no theoretical fan-out limitation. The pseudo codes of the P2P and AMC routing are shown in Algorithm 1.

## B. Existing Fully-unfolded/folded Mapping Schemes

There exist two schemes for mapping the convolution onto many-crossbar architecture: fully-unfolded and fully-folded one. The former is widely used in neuromorphic field (e.g. [15], [28], [33]), which unfolds all the memory and computation then transforms them to an FC layer with VMM operation for crossbar execution. Here we take 256×256 crossbar size as an example. As shown in Figure 4(a), for the simple FC layer with size of $256M \times 256N$, it can be completed by $M \times N$ crossbars (gray color) of 256 $256 \times$ size in VMM operation mode. All FunCs on the same row share the same input activations through AMC routing. Consequently, extra $N$ crossbars are used for accumulating the partial activations via VVA operation mode (red color). The packet propagation from VMM FunCs to VVA FunCs is carried by P2P routing. Here we just take divisible VMM size as an example, and it is easy to extend to undivisible cases by using extra crossbars for the residual computation. Based on the FC layer mapping, Conv and Pool layers are easy to map if we first transform them to an FC layer with VMM operation. As shown in Figure 4(b), a Conv layer with $C_1$ input FMs of $m_1 \times n_1$ size, $C_2$ output FMs of $m_2 \times n_2$ size, and $r \times r \times C_1 \times C_2$ weight kernel, can be converted to $m_2 \cdot n_2$ VMMs with size of $(r_2 C_1) \times C_2$ for each. Consequently, each VMM can be mapped in the same way as above FC layer. For Pool layer in Figure 4(c), because each neuron is only determined by its neighboring neurons in the corresponding input FM, it only requires $C \cdot m_2 \cdot n_2$ VB & Pool operations with size of $r_2 \times 1$. Here we use VB and Pool operations to replace the VMM and VVA operations in Conv layer. Note that, because $r_2$ is often smaller than 256, we can merge multiple $r_2 \times 1$ matrices together to occupy the whole crossbar.

The other extreme mapping scheme, fully-folded mapping [6], is presented in Figure 4(d). Here we focus on the Conv or Pool layer, because the FC layer has no data reuse and only fully-unfolded mapping is applicable. The fully-folded mapping only assigns the physical resources for one sliding window (red or blue box). Then it reuses these resources cycle by cycle until all the sliding window are finished. Note that in Pool layer, each output FM is just produced by its corresponding input FM rather than all input FMs in Conv layer, which is not illustrated in Figure 4(d) for clarity.

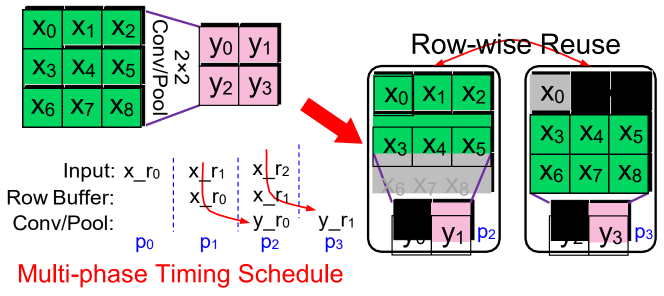


Fig. 5: SemiMap with folding along the row dimension and unfloding along the column dimension.

Overall, the fully-unfolded mapping consumes large amount of crossbar resources for high throughput, while the fully-folded mapping consumes plenty of clock cycles for saving resources. Consequently, the resulting speed and overhead are greatly imbalanced. To solve this imbalance, we propose a novel mapping framework (SemiMap) for efficient network deployment with several smart designs: row-wise folding and column-wise unfolding, FM slicing, multi-phase-per-step timing schedule, neuron reservation, and streaming pipeline. We will illustrate them one by one in the following subsections.

## C. SemiMap: Row-wise Folding and Column-wise Unfolding

In the fully-unfolded mapping, each cell has an independent physical space. While for the Conv layer, as well known, the activation and weight are greatly shared. This provides an opportunity for reusing the resources. Figure 5 presents our scheme of row-wise reuse, which takes a size of '3×3 input FM & 2×2 weight kernel & 2×2 output FM' as an example. In this case, we find the generation of each output row only requires two input rows, and the generation of consequent rows could reuse the same weight kernel. This promises the neuron multiplexing while remains the crossbar configuration unchanged.

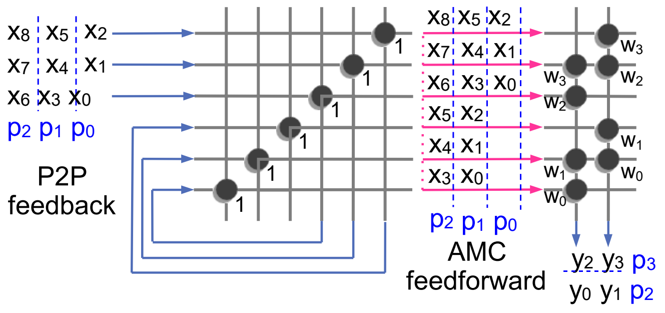


Fig. 6: Crossbar configuration through feedback and feedforward routing.

Specifically, we can just assign physical resources (a 6×2 crossbar and two neurons) for generating the first row '[$y_0$, $y_1$]'. Then at the next time phase, we can generate the second row '[$y_2$, $y_3$]' but still using the same resources. In this way, we fold the convolution along the row dimension for resource saving, but still unfold the computation and memory along the column dimension to maintain parallelism. Compared to the fully-unfolded mapping with one 9×4 crossbar and the fully-folded mapping with four sliding cycles, our SemiMap only consumes a 6×2 crossbar and two cycles (considering the pipeline to be introduced in Section III-G). Note that here we just take small kernel and FMs as an example. In reality with larger size, we also need to convert this basic computation within several rows to the FC-like placement in Figure 4(a).

Figure 6 shows its detailed implementation on the crossbar architecture. The input FM is injected into the row-buffer crossbar row by row at each phase, and the output is routed back (P2P) to the same crossbar as the input at next phase. The row-buffer crossbar uses diagonal identity weight matrix and VB operation. In fact, the VB operation will bypass the crossbar and the diagonal identity weight matrix is just visualized for better understanding. More importantly, the feedback connections are organized with a staggered row. In this way, the row-buffer crossbar is able to sequentially buffer and organize all the required rows within $K$ phases, where $K$ is the size of weight kernel. Here, we can get the first two rows '$x_0$-$x_5$' and the second two rows '$x_3$-$x_8$' at phase $p_1$ and $p_2$, respectively. These iterative inputs can be shared with the post computation crossbar through feedforward AMC routing. Then the consequent computation crossbar (VMM operation) can generate the output row '[$y_0$, $y_1$]' and '[$y_2$, $y_3$]' at phase $p_2$ and $p_3$, respectively. The weight kernel is copied for generating different elements in the same output FM row to maintain parallelism (unfolding), while they are reused among different rows for resource saving (folding). Specifically in this example at $p_2$, we have $y_0 = x_0w_0 + x_1w_1 + x_3w_2 + x_4w_3$ and $y_1 = x_1w_0 + x_2w_1 + x_4w_2 + x_5w_3$ with two copies of the weight kernel; while at $p_3$, the generation of $y_2 = x_3w_0 + x_4w_1 + x_6w_2 + x_7w_3$ or $y_3 = x_4w_0 + x_5w_1 + x_7w_2 + x_8w_3$ reuses the same weight kernel copy for the above generation of $y_0$ or $y_1$ at $p_2$, respectively. In this way, the row-dimension folding and column-dimension unfolding help achieve speed-overhead balance, and the row-by-row execution forms a seamless streaming dataflow.

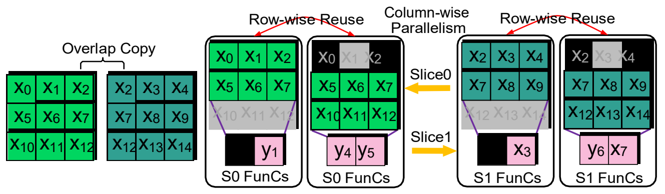


Fig. 7: FM slicing for large-size image.

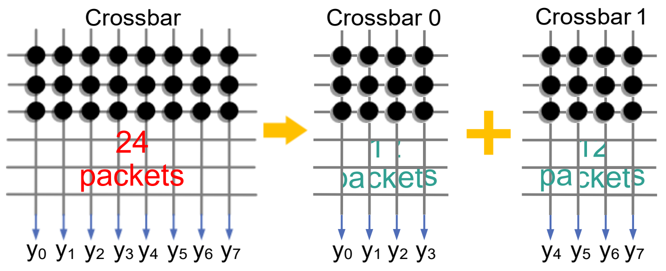


Fig. 8: Neuron reservation for routing guarantee in VVA operation.

## D. SemiMap: Feature Map Slicing

From the above example, we can see that multiple rows could be buffered by a row-buffer crossbar with feedback connections. However, if we have many FMs or FM with large width, it is impractical to accommodate all the required rows on one crossbar even if just generating one output row. For example, if the channel number is 512, FM width is 14, and weight kernel is 3×3, generating one output row requires 14×3×512=21504 input activations, which is often much larger than the fan-ins of one single crossbar. One usual way is to divide all input FMs into multiple groups for independent VMM operations and then accumulate the partial activations

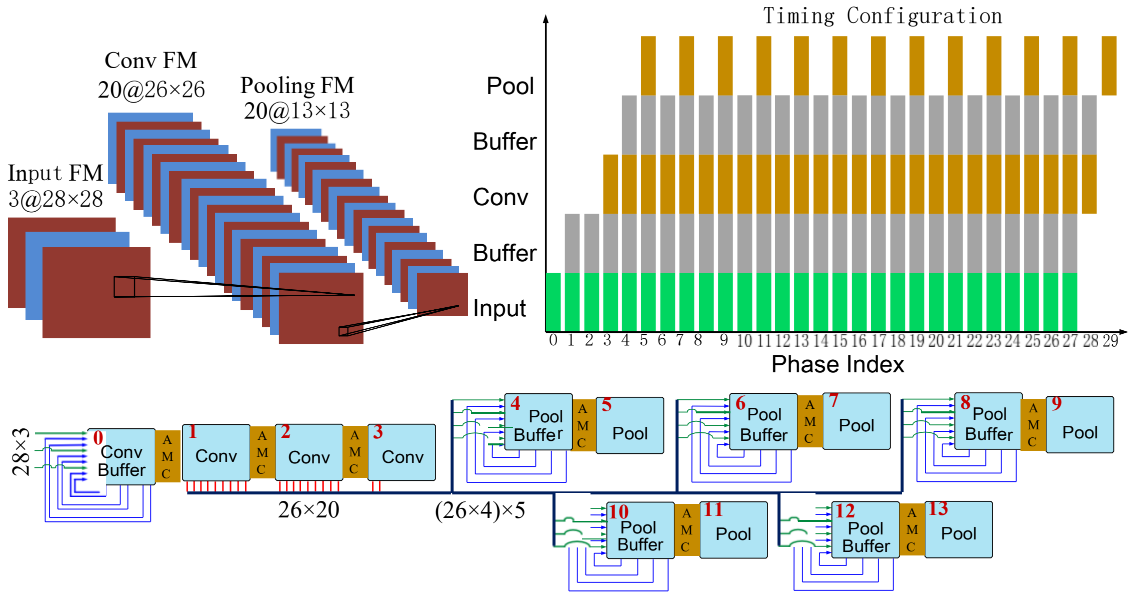


Fig. 9: A mapping example involving routing topology and timing configuration.

through VVA operations as shown in Figure 4(a). But at some extreme cases, such as FM width 224, it still exceeds the fan-in limitation of one single crossbar even if we only have one input FM. To address this issue, we propose an FM slicing scheme.

As shown in Figure 7, we take a similar example by just changing the FM size to ‘5×5 input FM & 4×4 output FM’. If our crossbar just has 6 fan-ins, which indicates it cannot hold two input rows with 10 neurons at the same time. In this case, we slice the FM along the column dimension with three neurons per row in each slice. Note that because of the convolution overlap, we need to copy the overlapped neurons between two adjacent slices using the Out_Copy operation. Then we can map the two slices onto independent crossbars for parallel execution. For every slice, we can do the same row-wise streaming reuse like that in Section III-C.

## E. SemiMap: Neuron Reservation

In large scale networks, especially when the number of input FMs is large (e.g. 256, 512, etc.), it is still impossible to put all inputs onto a single crossbar even if we use the slicing scheme to reduce the FM width of each slice to only 3 (the minimum value limited by the weight kernel size such as 3×3). Therefore, we have to add extra crossbars for partial activation accumulation through VVA operation. As shown in Figure 4(a), although it is presented for the FC layer mapping, it’s still required in SemiMap for the Conv layer. We can see that each VMM FunC only receives its own input FMs, but each VVA FunC needs to reduce all the partial activation vectors to be one complete vector. So the P2P routing burden is heavy in the VVA FunCs. Here we propose a neuron reservation method to alleviate this heavy traffic.

Figure 8 presents a simplified example to explain how it works. The original VVA crossbar needs to reduce 3 partial activation vectors with length of 8 for each to one complete vector $y_0$-$y_7$. In this case, the total routing packets injected to this crossbar at one phase is 8×3=24. If the peak routing capability of one FunC during one phase is receiving 15 packets, the required packets will exceed this constraint. Then, we activate the neuron reservation method, which uses multiple (here is two) crossbars to finish the same task and each crossbar only shoulders 4×3=12 packets. This reduction meets the routing constraint. Although we leave some neurons along with the corresponding crossbar columns underutilized in every crossbar and use more crossbars, the routing capability is well guaranteed. Since the VVA FunCs are much less than the VMM ones, this extra cost is negligible (to be shown in Figure 12).

## F. SemiMap: Multi-phase-per-step Timing Schedule

In fact, the above row-wise streaming mapping cannot be implemented without the support of compatible timing schedule. In conventional neuromorphic architecture [1], the minimum timing unit is the the time step. After the configuration initialization, the chip runs step by step with identical operations during every step. However, as shown in Figure 5, we expect the compute and transformation engine to execute different pattern at each phase during an intra-frame period. For instance, we want the Conv or Pool FunCs to start the effective compute and transformation at $p_2$ and $p_3$ phases, but disable them at $p_0$ and $p_1$ phases. Otherwise, we will obtain incorrect output if we start the calculation before the organization of required rows from the row buffer FunC. To realize this non-identical execution pattern, we propose a multi-phase-per-step timing schedule.

As illustrated in Figure 9, we use a Conv-Pool layer pair to demonstrate how to implement this timing design. Here the layer structure is ‘28×28×3-20C3P0S1-MP2’, where the 28×28×3 indicates there are 3 input FMs with size of 28×28, 20C3 denotes 20 output FMs with 3×3 weight kernels, P or S denotes the padding or stride value, respectively, and MP2 means max pooling with 2×2 pooling window. Here the Conv stride is 1 and Pool stride is 2, and both of them have no padding. Generating one output row requires 3 input rows, i.e. 28×3×3=252<256 neuronal activations. Therefore, only one crossbar for row buffer is enough here. Because each crossbar can fan out 9 output FMs (26×9=234<256), it requires 3 Conv crossbars to generate 20 output FMs. Then the 20 output FMs are divided into 5 groups for pooling operation. In each group, there are 4 FMs, i.e. totally 26×2×4=208<256 neuronal activations after considering that this pooling requires 2 input rows. As mentioned in Section II-A, the FMs are decoupled in Pool layer, so each row buffer crossbar just requires another one crossbar for its consequent pooling operations. Thus, we totally use 14 FunCs for this Conv-Pool layer pair.

The multi-phase configuration within one step is shown in the top right of Figure 9. The input FMs are injected row by row (0-27 phase for 28 rows). Then the Conv buffer crossbar starts the row buffering from the second phase. It requires 3 phases to organize the first 3 rows for one convolution operation, thus the Conv crossbars can start its compute and transformation engine from the next phase (i.e. from phase 3). The Pool layer follows the similar pattern with Conv layer. The major difference is that it enables its compute and transformation engine every 2 phases. This is because the stride of pooling operation is 2 (i.e. no overlap like that in Conv layer). In a nutshell, the overall timing sequence acts like a streaming dataflow with inter-crossbar relay.

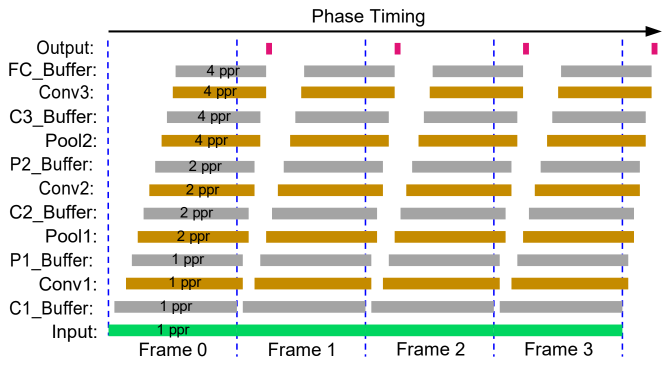


Fig. 10: Inter-frame periodical pipeline. Here ‘ppr’ denotes phases per row.

## G. *SemiMap: Inter-frame Periodical Pipeline*

After introducing the implementation details for a single layer, now we present the timing sequence of a whole network as shown in Figure 1 to gain the big picture. Figure 10 shows the timing occupation of every layer within one frame and among continuous frames. Here the ‘ppr’ represents phases per row, i.e. how many phases are required to generate one output row. Two observations we can get: (1) The ppr will double every time passing a Pool layer (here we use 2×2 pooling window with stride 2), i.e. its activity becomes sparser as layer propagates; (2) The inter-frame timing sequence presents as a periodical pipeline. The throughput is only determined by the height of the input image (number of rows), and it is decoupled with the network depth. This feature is distinct from the conventional accelerators [34]–[36] whose performance is mainly determined by the whole model size.

## IV. Results

### A. *Experimental Setup*

In order to reduce the development period and save the fabrication cost, we use off-the-shelf SRAM array with extra multipliers and accumulators to simulate the crossbar-based compute engine (i.e. virtual crossbar). Note that our mapping framework is also applicable to other crossbars (virtual or physical), as aforementioned in Section II-B. We fabricate a chip in UMC 28nm HLP CMOS process to implement the designed many-crossbar architecture. Considering the fabrication cost, we only integrate 156 FunCs onto one single chip. Figure 11 shows the IC layout and real chip picture. At 300 MHz clock frequency, the chip runs (Chip_busy=1) only within the first 16.8 $\mu s$ during each phase to complete all the computations, which reflects the minimum phase latency for guaranteeing the running correctness. Then we develop a mapping compiler in Matlab for network partition and resource allocation, and a C++ cycle-accurate simulator for the hardware simulation of large-scale networks. For all the experiments in this paper, the architecture configuration in the simulator is listed in Table II. The power estimation is based on the measured data of 1.95~6.29 mW per FunC in different operation modes (Table I) or idle mode. In this paper, we focus more on the mapping methodology, not the specific hardware design. With this concern, we don’t consider the inter-chip communication cost in our simulations, which can be optimized by techniques such as using ultra-high communication bus [37] (higher speed) or integrating more FunCs onto one single chip [1] (lower power & higher speed).

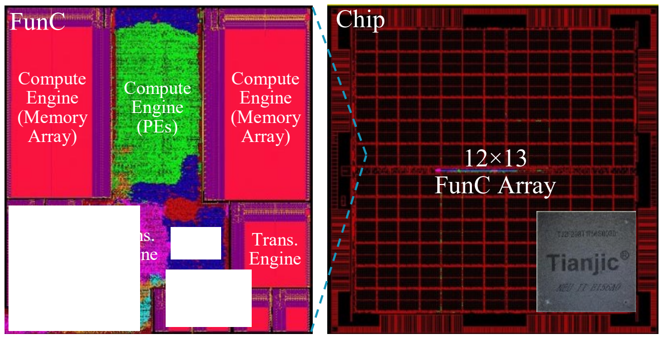


Fig. 11: Chip layout and physical picture.

TABLE II: Architecture configuration in the simulator.

| FunCs per Chip | Memory Crossbar | Data Precision |
|---|---|---|
| 12×13 | 256×256 (SRAM) | 8b-W/8b-A |
| **Packet Length** | **Clock Frequency** | **Phase Latency** |
| 42 bits | 300 MHz | 16.8 $\mu s$ |

Regarding the evaluation network models, we use LeNet-variant [38] on MNIST [39], VGG8 [38], [40], on CIFAR10 [27], and AlexNet [29]/VGG16 [30]/ResNet18 [31] on ImageNet [32] as benchmarks. All the mapping schemes including fully-unfolded, fully-folded, and SemiMap are completed in our mapping compiler. The resource cost is directly obtained from the mapping compiler, and the throughput is achieved from the cycle-accurate simulator. Regarding the baseline hardware platforms for lateral comparison, we use NVIDIA GPU (Titan Xp) as well as several existing accelerators including DRISA [41], Eyeriss [36], and DNA [42].

## B. Layer Analysis

In this section, we use a single layer for a detailed analysis. The layer is the conv2-2 layer from VGG16 network, the structure of which is ‘112×112×128-128C3P1S1’.

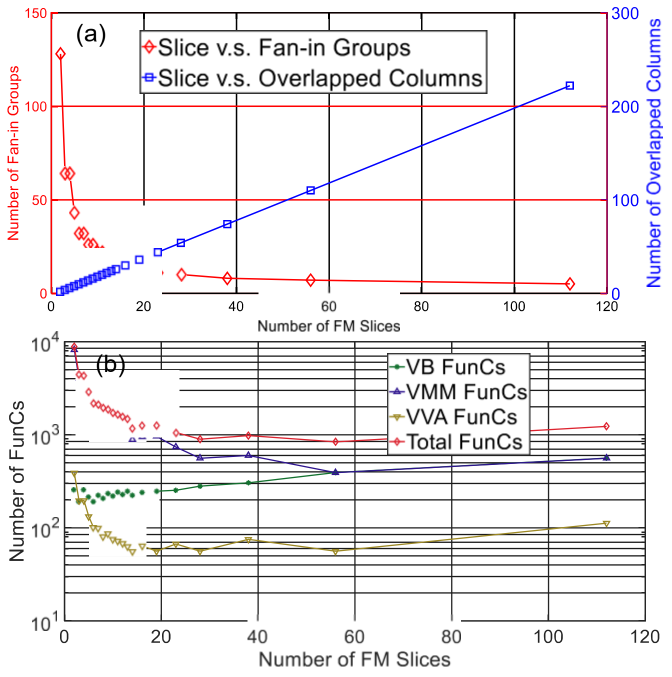


Fig. 12: Profiling of the SemiMap results on one single Conv layer: (a) relationship between the number of FM slices and fan-in groups or overlapped columns; (b) relationship between the number of FM slices and resource overhead.

Figure 12 presents the the mapping results of this layer using the proposed SemiMap. From Figure 12(a), we can see that the number of fan-in groups decreases with more FM slices. Because more slices make the width of each FM slice smaller, hence, each crossbar can accommodate more input FMs (i.e. less fan-in groups). However, more slices will generate more overlapped columns between adjacent slices, which requires linearly increasing copy neurons. From Figure 12(b), we can see a trade-off between the slices and resource overhead (i.e. number of FunCs). Too few slices generate wider FM slices and many fan-in groups, that brings lower crossbar utilization and larger crossbar amount. Meanwhile, too many slices require more crossbars for row buffer since each slice needs independent buffers. This will also increase the resource overhead of this layer. In addition, the number of copy neurons from previous layer increases due to more overlapped columns. Another observation is that the VVA FunCs for accumulating the partial activations occupy the least fraction because they use the crossbar itself with larger memory capacity for data stash, not the small input buffer in VMM or VB operation mode.

TABLE III: Layer mapping with three schemes.

| Scheme | Number of FunCs | | | | Phases |
|---|---|---|---|---|---|
| | VB | VMM | VVA | Total | |
| Fully-unfolded | – | 62720 | 12544 | 75264 | 1 |
| Fully-folded | – | 5 | 1 | 6 | 12544 |
| **SemiMap** | 224 | 896 | 56 | 1176 | 115 |

Table III shows the overall comparison among three mapping schemes: fully-unfolded, fully-folded, and SemiMap. Note that we ignore the copy neurons for the copy of overlapped input activations. Fully-unfolded mapping can do all things in only one phase, however, at the cost of huge resource overhead, more than $7.5\times10^4$ FunCs. In stark contrast, the fully-folded one saves resources to only several FunCs, while, consuming more than $1.2\times10^4$ timing phases. This speed-overhead imbalance is probably not acceptable in practical applications. By using the proposed SemiMap, we can execute this layer within 115 phases, and consumes only 1176 FunCs. This reduces 64x resource overhead and 109x time cost compared to the fully-unfolded and fully-folded mapping scheme, respectively.

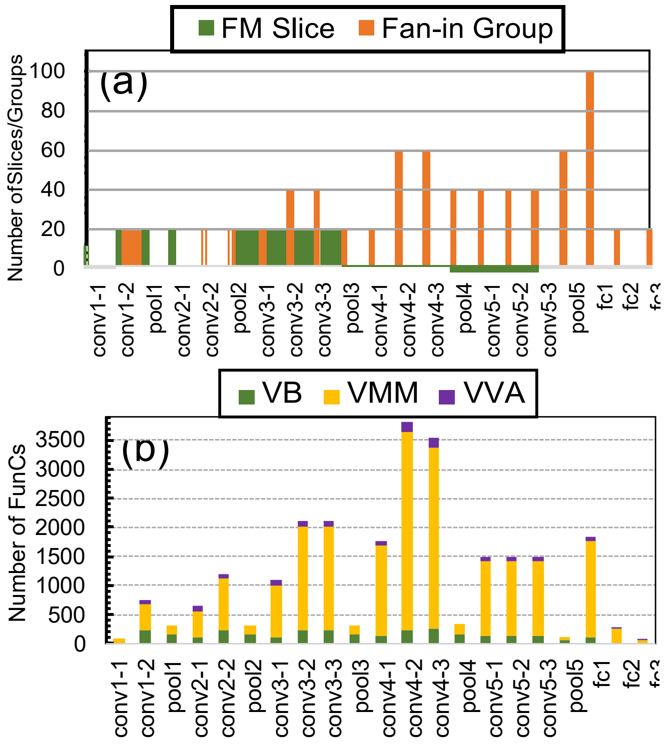


Fig. 13: Profiling of the SemiMap results on VGG16 network.

## C. Network Evaluation

Based on the single layer analysis, we give the mapping results on a complete network, i.e. VGG16. As shown in Figure 13(a), for the first few layers, more slices are usually

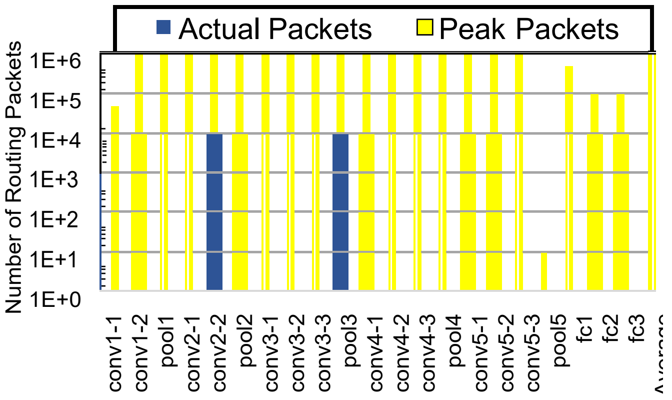


Fig. 14: Inter-layer routing redundancy.

required to split the large-size FMs. Reversely, the fan-in groups increase as the layer propagates, which is mainly caused by the increasing number of FMs. In observing the resource overhead in Figure 13(b), we can get the consistent conclusion with Figure 12, that the VMM FunCs occupy the most overhead and the VVA FunCs are the least. The Conv layers usually consume more resources than the FC layers, which is different from that of fully-folded mapping scheme [6]. Furthermore, the Conv layers with more FMs usually consume more crossbars. For example, the conv4 layer family with 512 FMs occupies the most resources. The less overhead of the conv5 layers is caused by the smaller FM size after a pooling layer (pool4).

For many-crossbar architectures, the inter-FunC communication is the key for the activation movement and the overall performance. Figure 14 shows the routing packets between adjacent layers, from the output FunCs of previous layer to the input FunCs of current layer. In Conv layer, the inter-layer receiver is the VB FunCs for buffering rows, and in FC layer, the receiver is VMM FunCs. We can see that the actual routing traffic doesn't reach the peak capability. This 86x routing redundancy guarantees the smooth communication.

However, as shown in Figure 4(a), the reduce FunCs in VVA operation mode has significantly increased traffic burden since they have to receive the partial activations from all the VMM FunCs. As mentioned in Section III-E, we design a neuron reservation technique to optimize the routing capability. As shown in Figure 15, first, we make full use of the crossbar neurons. But in this case, the actual packets in many layers exceed the peak routing capability. This may probably cause communication failure, i.e. cannot complete the packet transmission during one phase, thus cause system crash. After leveraging the neuron reservation optimization, we utilize less neurons in each VVA crossbar and keep some neurons along with the corresponding crossbar columns underutilized. Although this increases the number of VVA FunCs slightly, we are able to guarantee the routing performance (with 1.14x routing redundancy on VGG16). It's worthy noting that, from Figure 13, the VVA FunCs are the least part among the overall resource overhead, so the slightly increased resource overhead by using this routing optimization is negligible.

## *D. Optimized Speed-Overhead Trade-off*

In this section, we will provide more overall evaluation on more networks, and compare with other state-of-the-art platforms. Figure 16 shows the resource overhead comparison between the conventional fully-unfolded mapping and our SemiMap. We can save many VMM FunCs for Conv layers, VB FunCs for Pool layers (as well as other VB FunCs for row buffer or overlapped neuron copy), and VVA FunCs. In general, we are able to reduce 10x-36x resource overhead. Because the proposed SemiMap is mainly designed for Conv structure, there is little difference between fully-unfolded mapping and SemiMap on FC layers.

On the speed side, Figure 17 shows the phase comparison between the conventional fully-folded mapping and the proposed SemiMap. We don't consider the possible inter-layer pipeline in fully-folded mapping, which means that the processing of all the layers is serial. We are able to achieve much less phases (23x-462x) on these benchmarks. Interestingly, combined with the inter-frame pipeline mentioned in Section III-G, the throughput of SemiMap is only determined by the height of input image (number of rows). This is different from most existing CNN accelerators whose throughput is mainly determined by the model size. Note that the very small differences in SemiMap among AlexNet, VGG16 and ResNet18 on ImageNet are caused by the different padding values.

At last, we compare the throughput of our SemiMap on the proposed many-crossbar architecture with that of GPU and several accelerators. Note that here we only provide coarse comparisons, because it's difficult to make it vary fair on these different architectures if arguing on low-level details. As shown in Figure 18, we can achieve 1.5x-2.8x speedup over GPU on the datasets with small images, and slight speedup (1.1x-1.4x) on ImageNet (except for AlexNet). As aforementioned, the throughput of SemiMap is mainly determined by the input image size rather than the overall model size. So the reason why it performs a bit worse on AlexNet is because its model size is relatively small but the image size is large, which makes it less friendly to our SemiMap framework on crossbars.

Table IV shows the throughput comparison with several published CNN accelerators. Benefit from the high bandwidth of processing-in-memory architecture, DRISA [41] achieves high throughput, but we still present a slight speedup (1.3x) averagely. The throughput of other two accelerators, Eyeriss [36] and DNA [42], demonstrates strong dependency on the model size (although DNA doesn't show the results on VGG16). We averagely achieve 14.7x and 2x speedup over Eyeriss and DNA, respectively.

TABLE IV: Throughput comparison with existing accelerators.

| Network | DRISA [41] | Eyeriss [36] | DNA [42] | **SemiMap** |
|---|---|---|---|---|
| AlexNetConv | 358 | 35 | 136 | 262 |
| AlexNet | 352 | N.A. | 126 | 262 |
| VGG16Conv | 46 | 0.7 | N.A. | 263 |
| VGG16 | 45 | N.A. | N.A. | 263 |
| Average | 200.3 | 17.9 | 131.0 | 262.5 |

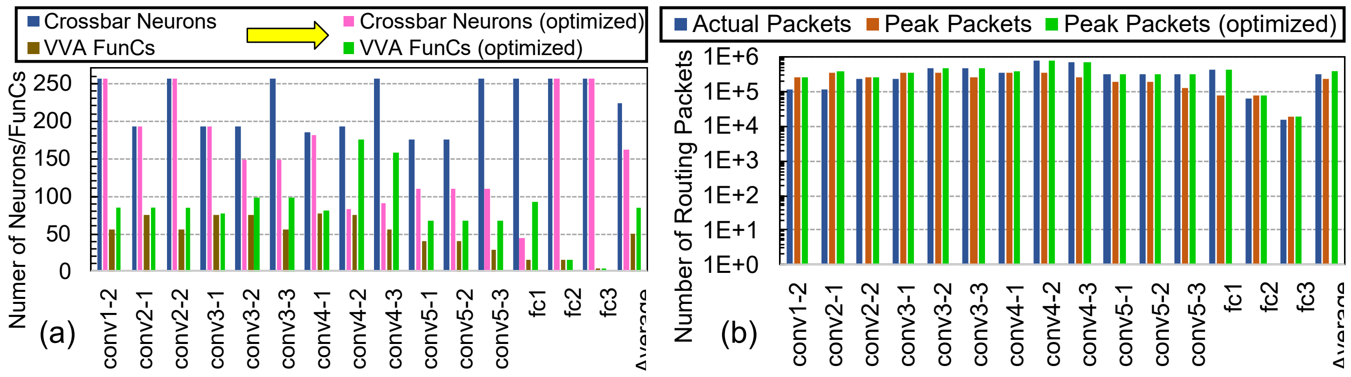


Fig. 15: VVA routing optimized by neuron reservation.

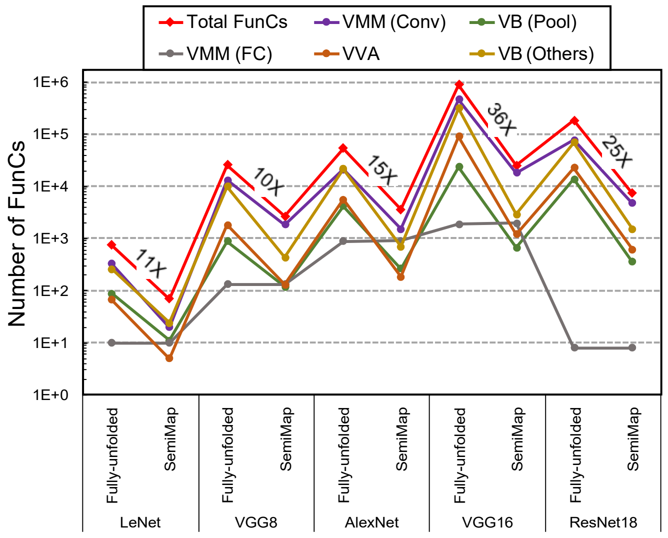


Fig. 16: FunC consumption comparison between fully-unfolded mapping and SemiMap.

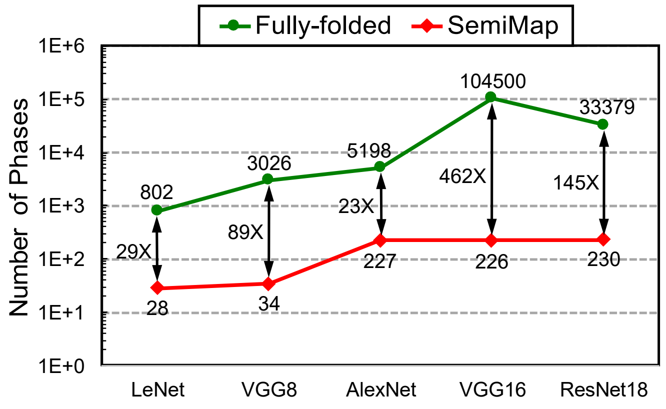


Fig. 17: Phase consumption comparison between fully-folded mapping and SemiMap.

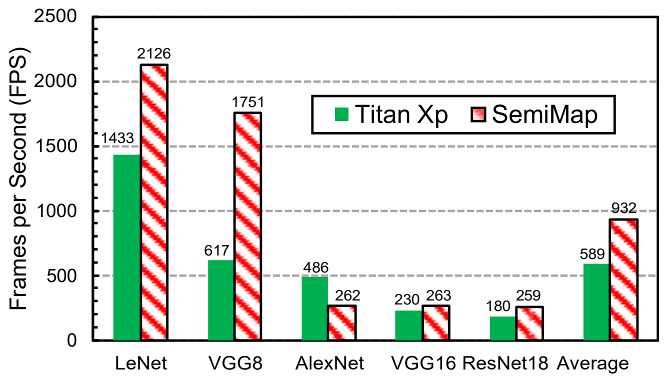


Fig. 18: Throughput comparison between GPU and this work on many-crossbar architecture with SemiMap.

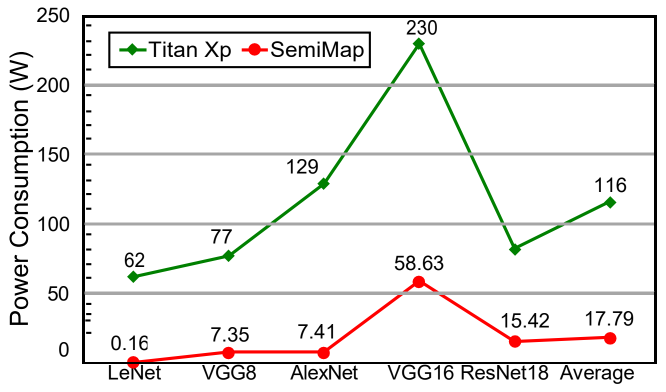


Fig. 19: Power comparison between GPU and this work on many-crossbar architecture with SemiMap.

Besides the throughput evaluation, we also provide a coarse power comparison with GPU, as shown in Figure 19. An average 6.5x improvement of power efficiency is achieved. Note that the computation of zero rows on the crossbar with VMM operation is skipped for further energy reduction.

In a nutshell, by implementing the proposed SemiMap scheme on the many-crossbar architecture, it is able to approach a good balance between the speed and resource overhead. Regarding the speed, even if we sacrifice the performance to some extent for resource saving, we still demonstrate higher throughput compared to existing platforms; regarding the resource, the reduced overhead brings much lower power consumption compared to GPU.

## V. Conclusion and Discussion

In this work, we propose a SemiMap framework for balancing the execution speed and resource overhead on the widely used crossbar architecture. Through row-wise folding and column-wise unfolding, it is able to achieve the reduced overhead and maintained parallelism in the meantime. FM slicing scheme allows the processing of large-size image, multi-phase-per-step timing schedule enables complex intra- and inter-frame streaming pipeline, and AMC routing and routing-aware neuron reservation optimize the communication capability. To validate the mapping methodology, a many-crossbar chip is designed and fabricated. Based on the measurement data, a mapping compiler and a cycle-accurate simulator are developed for the simulation of large-scale networks. Testing over several CNN benchmarks, SemiMap is able to reduce the crossbar overhead up to 36x and accelerate the execution phase up to 462x. In the coarse lateral comparisons, it performs up to 2.8x and 14.7x throughput improvement compared to state-of-the-art GPU and accelerators, respectively, as well as much less power consumption than GPU.

Because CNNs have been proven to be more powerful than MLP networks, the crossbar architecture must solve the challenge of speed-overhead imbalance to support CNNs more efficiently. In this work, we address this issue well by optimizing the higher-level mapping scheme, which provides a new way to improve the performance. Our implementation using off-the-shelf SRAM array and extra PEs to mimic the crossbar behavior is just for cost saving, and in fact, the proposed SemiMap can be easily extended to other crossbar architectures, such as the emerging devices with in-memory computing [5]–[9], [21]–[26]. Furthermore, since the convolution operation is symmetric in both the row and column dimensions. The row-driven mapping and timing schedule in this paper can be easily extended to a column-driven version. One disadvantage of this SemiMap scheme is that the crossbars cannot be fully utilized along the temporal dimension. Specifically, as shown in Figure 10, each FunC occupies only a part of the periodical duration. This insufficient utilization becomes more severe in the last few layers with large ppr (i.e. more sparser). In the idle state, the peripheral circuits within the FunC still consume power, such as memories, router, clock tree, and the leakage. This will decrease the overall power efficiency to some extent. But anyway, regarding the throughput and resource overhead, we can achieve a good balance. The temporal utilization issue is one of our future works. Another problem deserves investigation is the spatial utilization on the crossbar. In contrast to the naturally high utilization of FC layer, Conv layer only occupies a fraction of the crossbar cells (many of them are zero values) due to the sliding operations. One promising direction is to study the crossbar-aware sparsification like that in [43].

## References

[1] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura, *et al.*, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, 2014.

[2] S. B. Furber, F. Galluppi, S. Temple, and L. A. Plana, "The spinnaker project," *Proceedings of the IEEE*, vol. 102, no. 5, pp. 652–665, 2014.

[3] L. Shi, J. Pei, N. Deng, D. Wang, L. Deng, Y. Wang, Y. Zhang, F. Chen, M. Zhao, S. Song, F. Zeng, G. Li, H. Li, and C. Ma, "Development of a neuromorphic computing system," in *2015 IEEE International Electron Devices Meeting (IEDM)*, pp. 4.3.1–4.3.4, Dec 2015.

[4] D. Garbin, O. Bichler, E. Vianello, Q. Rafhay, C. Gamrat, L. Perniola, G. Ghibaudo, and B. DeSalvo, "Variability-tolerant convolutional neural network for pattern recognition applications based on oxram synapses," in *Electron Devices Meeting (IEDM), 2014 IEEE International*, pp. 28–4, IEEE, 2014.

[5] M. Prezioso, F. Merrikh-Bayat, B. Hoskins, G. Adam, K. K. Likharev, and D. B. Strukov, "Training and operation of an integrated neuromorphic network based on metal-oxide memristors," *Nature*, vol. 521, no. 7550, p. 61, 2015.

[6] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "Prime: a novel processing-in-memory architecture for neural network computation in reram-based main memory," in *ACM SIGARCH Computer Architecture News*, vol. 44, pp. 27–39, IEEE Press, 2016.

[7] G. W. Burr, R. M. Shelby, S. Sidler, C. Di Nolfo, J. Jang, I. Boybat, R. S. Shenoy, P. Narayanan, K. Virwani, E. U. Giacometti, *et al.*, "Experimental demonstration and tolerancing of a large-scale neural network (165 000 synapses) using phase-change memory as the synaptic weight element," *IEEE Transactions on Electron Devices*, vol. 62, no. 11, pp. 3498–3507, 2015.

[8] J. Zhang, Z. Wang, and N. Verma, "In-memory computation of a machine-learning classifier in a standard 6t sram array," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 4, pp. 915–924, 2017.

[9] X. Guo, F. M. Bayat, M. Bavandpour, M. Klachko, M. Mahmoodi, M. Prezioso, K. Likharev, and D. Strukov, "Fast, energy-efficient, robust, and reproducible mixed-signal neuromorphic classifier based on embedded nor flash memory technology," in *Electron Devices Meeting (IEDM), 2017 IEEE International*, pp. 6–5, IEEE, 2017.

[10] W. Maass, "Networks of spiking neurons: the third generation of neural network models," *Neural networks*, vol. 10, no. 9, pp. 1659–1671, 1997.

[11] S. Ghosh-Dastidar and H. Adeli, "Spiking neural networks," *International journal of neural systems*, vol. 19, no. 04, pp. 295–308, 2009.

[12] J. Schemmel, D. Briiderle, A. Griibl, M. Hock, K. Meier, and S. Millner, "A wafer-scale neuromorphic hardware system for large-scale neural modeling," in *Circuits and systems (ISCAS), proceedings of 2010 IEEE international symposium on*, pp. 1947–1950, IEEE, 2010.

[13] B. V. Benjamin, P. Gao, E. McQuinn, S. Choudhary, A. R. Chandrasekaran, J.-M. Bussat, R. Alvarez-Icaza, J. V. Arthur, P. A. Merolla, and K. Boahen, "Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations," *Proceedings of the IEEE*, vol. 102, no. 5, pp. 699–716, 2014.

[14] J. Hasler and H. B. Marr, "Finding a roadmap to achieve large neuromorphic hardware systems," *Frontiers in neuroscience*, vol. 7, p. 118, 2013.

[15] F. Akopyan, J. Sawada, A. Cassidy, R. Alvarez-Icaza, J. Arthur, P. Merolla, N. Imam, Y. Nakamura, P. Datta, G.-J. Nam, *et al.*, "Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 10, pp. 1537–1557, 2015.

[16] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, *et al.*, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018.

[17] P. Merolla, J. Arthur, R. Alvarez, J.-M. Bussat, and K. Boahen, "A multicast tree router for multichip neuromorphic systems," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 61, no. 3, pp. 820–833, 2014.

[18] E. Painkras, L. A. Plana, J. Garside, S. Temple, F. Galluppi, C. Patterson, D. R. Lester, A. D. Brown, and S. B. Furber, "Spinnaker: A 1-w 18-core system-on-chip for massively-parallel neural network simulation," *IEEE Journal of Solid-State Circuits*, vol. 48, no. 8, pp. 1943–1953, 2013.

[19] L. Deng, G. Li, N. Deng, D. Wang, Z. Zhang, W. He, H. Li, J. Pei, and L. Shi, "Complex learning in bio-plausible memristive networks," *Scientific reports*, vol. 5, p. 10684, 2015.

[20] G. Li, L. Deng, D. Wang, W. Wang, F. Zeng, Z. Zhang, H. Li, S. Song, J. Pei, and L. Shi, "Hierarchical chunking of sequential memory on neuromorphic architecture with reduced synaptic plasticity," *Frontiers in computational neuroscience*, vol. 10, p. 136, 2016.

[21] Y. Long, E. M. Jung, J. Kung, and S. Mukhopadhyay, "Reram crossbar based recurrent neural network for human activity detection," in *Neural Networks (IJCNN), 2016 International Joint Conference on*, pp. 939–946, IEEE, 2016.

[22] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar, "Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 14–26, 2016.

[23] T. Tang, L. Xia, B. Li, Y. Wang, and H. Yang, "Binary convolutional neural network on rram," in *Design Automation Conference (ASP-DAC), 2017 22nd Asia and South Pacific*, pp. 782–787, IEEE, 2017.

[24] L. Song, X. Qian, H. Li, and Y. Chen, "Pipelayer: A pipelined reram-based accelerator for deep learning," in *High Performance Computer Architecture (HPCA), 2017 IEEE International Symposium on*, pp. 541–552, IEEE, 2017.

[25] O. Bichler, M. Suri, D. Querlioz, D. Vuillaume, B. DeSalvo, and C. Gamrat, "Visual pattern extraction using energy-efficient 2-pcm synapse neuromorphic architecture," *IEEE Transactions on Electron Devices*, vol. 59, no. 8, pp. 2206–2214, 2012.

[26] D. Fan and S. Angizi, "Energy efficient in-memory binary deep neural network accelerator with dual-mode sot-mram," in *2017 IEEE 35th International Conference on Computer Design (ICCD)*, pp. 609–612, IEEE, 2017.

[27] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," tech. rep., Citeseer, 2009.

[28] S. K. Esser, P. A. Merolla, J. V. Arthur, A. S. Cassidy, R. Appuswamy, A. Andreopoulos, D. J. Berg, J. L. McKinstry, T. Melano, D. R. Barch, C. di Nolfo, P. Datta, A. Amir, B. Taba, M. D. Flickner, and D. S. Modha, "Convolutional networks for fast, energy-efficient neuromorphic computing," *Proceedings of the National Academy of Sciences*, vol. 113, no. 41, pp. 11441–11446, 2016.

[29] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[30] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[31] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

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

[33] Y. Ji, Y. Zhang, W. Chen, and Y. Xie, "Bridge the gap between neural networks and neuromorphic hardware with a neural network compiler," in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 448–460, ACM, 2018.

[34] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, *et al.*, "Dadiannao: A machine-learning supercomputer," in *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 609–622, IEEE Computer Society, 2014.

[35] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "Eie: efficient inference engine on compressed deep neural network," in *Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on*, pp. 243–254, IEEE, 2016.

[36] 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.

[37] A. Roshan-Zamir, O. Elhadidy, H.-W. Yang, and S. Palermo, "A reconfigurable 16/32 gb/s dual-mode nrz/pam4 serdes in 65-nm cmos," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 9, pp. 2430–2447, 2017.

[38] L. Deng, P. Jiao, J. Pei, Z. Wu, and G. Li, "Gxnor-net: Training deep neural networks with ternary weights and activations without full-precision memory under a unified discretization framework," *Neural Networks*, vol. 100, pp. 49–58, 2018.

[39] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[40] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *Advances in neural information processing systems*, pp. 3123–3131, 2015.

*[41]* S. Li, D. Niu, K. T. Malladi, H. Zheng, B. Brennan, and Y. Xie, "Drisa: A dram-based reconfigurable in-situ accelerator," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 288–301, ACM, 2017.

[42] F. Tu, S. Yin, P. Ouyang, S. Tang, L. Liu, and S. Wei, "Deep convolutional neural network architecture with reconfigurable computation patterns," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 8, pp. 2220–2233, 2017.

[43] L. Liang, L. Deng, Y. Zeng, X. Hu, Y. Ji, X. Ma, G. Li, and Y. Xie, "Crossbar-aware neural network pruning," *IEEE Access*, vol. 6, pp. 58324–58337, 2018.

**Lei Deng** received his B.E. degree and Ph.D. degree from University of Science and Technology of China, Hefei, China, and Tsinghua University, Beijing, China, in 2012 and 2017, respectively. He is currently a Postdoc at Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA, USA. Dr. Deng is a Guest Associate Editor for Frontiers in Neuroscience. His research interests include computer architecture, machine learning, brain-inspired computing, tensor analysis, and complex systems.

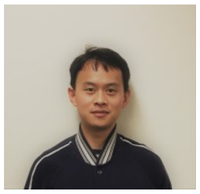
**Ling Liang** received the B.E. degree from Beijing University of Posts and Telecommunications, Beijing, China, in 2015, and M.S. degree from University of Southern California, CA, USA, in 2017. He is currently pursuing the Ph.D. degree at Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA, USA. His current research interests include machine learning security and computer architecture.

**Guanrui Wang** received the B.E. degree in electronic engineering from Jilin University, Jilin China, in 2016. He is currently pursuing the Ph.D. degree in instrumentation science and technology at Tsinghua University, Beijing, China. His current research interests include neuromorphic chips, computer architecture, and network-on-chip systems.

**Liang Chang** received the M.S. degree in microelectronics from Beihang University, Beijing, China, in 2014. He is currently pursuing the Ph.D. degree in spintronics, at the Fert Beijing Institute, BDBC, and the School of Electronic Information and Engineering, Beihang University. His research interests include reconfigurable circuit design and advanced computer architectures based on emerging non-volatile memory devices.

**Xing Hu** received the B.S. degree from Huazhong University of Science and Technology, Wuhan, China, and Ph.D. degree from University of Chinese Academy of Sciences, Beijing, China, in 2009 and 2014, respectively. She is currently a Postdoc at Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA, USA. Her current research interests include emerging memory system and domain-specific hardware computing.

**Xin Ma** received his B.S. and Ph.D degree in Physics from University of Science and Technology of China, Hefei, China, in 2009 and The College of William and Mary, VA, USA, in 2014 respectively. He is currently a Postdoc at Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA,USA. His research interests include designing in-memory computing with emerging non-volatile memory technologies.



**Liu Liu** received the B.S. degree and Ph.D degree from University of Electronic Science and Technology of China, Chengdu, China, and University of California, Santa Barbara, CA, USA, in 2013 and 2015, respectively. He is currently pursuing the Ph.D. degree with the Department of Computer Science, University of California, Santa Barbara, CA, USA. His research interests include deep learning, computer architecture, and emerging non-volatile memory.

**Jing Pei** received his B.E. and M.E. degrees in instrument science and technology from Tsinghua University, Beijing, China, in 1987 and 1989, respectively. He worked for Tsinghua University since 1990. He is currently an Associate Professor with the Department of Precision Instrument, Tsinghua University, Beijing, China. He has published over 50 papers in journals and conferences and obtained over 30 invention patents. He won once second prize of national invention award and once second prize of national scientific and technological progress award. His research interests include optical information storage system, optical signal processing, and recently, he focuses on specialized chip for brain-inspired computing based on neuromorphic engineering.

**Guoqi Li** received the B.E. degree from the Xian University of Technology, Xian, China, in 2004, the M.E. degree from Xian Jiaotong University, Xian, China, in 2007, and the Ph.D. degree from Nanyang Technological University, Singapore, in 2011. He was a Scientist with Data Storage Institute and the Institute of High Performance Computing, Agency for Science, Technology and Research (ASTAR), Singapore, from 2011 to 2014. He is currently an Associate Professor with the Department of Precision Instrument, Tsinghua University, Beijing, China. He has published over 70 journal and conference papers. His current research interests include brain inspired computing, complex systems, neuromorphic computing, machine learning, and system identification.

**Yuan Xie** received his Ph.D. degrees from Electrical Engineering Department, Princeton University, Princeton, NJ, USA in 2002. He was with IBM, Armonk, NY, USA, from 2002 to 2003, and AMD Research China Lab, Beijing, China, from 2012 to 2013. He was a Professor with Pennsylvania State University, State College, PA, USA, from 2003 to 2014. He is currently a Professor with the Department of Electrical and Computer Engineering, University of California at Santa Barbara, Santa Barbara, CA, USA.

Dr. Xie is an expert in computer architecture who has been inducted to ISCA/MICRO/HPCA Hall of Fame. He has been an IEEE Fellow since 2015. He served as the TPC Chair for HPCA 2018 and he is Editor-in-Chief for ACM Journal on Emerging Technologies in Computing Systems (JETC), Senior Associate Editor (SAE) for ACM Transactions on Design Automations for Electronics Systems (TODAES), and Associate Editor for IEEE Transactions on Computers (TC). His current research interests include computer architecture, Electronic Design Automation, and VLSI design.