A High Energy Efficient Reconfigurable Hybrid Neural Network Processor for Deep Learning Applications

Shouyi Yin, Peng Ouyang, Shibin Tang, Fengbin Tu, Xiudong Li, Shixuan Zheng, Tianyi Lu, Jiangyuan Gu, Leibo Liu, and Shaojun Wei

***Abstract*— Hybrid neural networks (hybrid-NNs) have been widely used and brought new challenges to NN processors. Thinker is an energy efficient reconfigurable hybrid-NN processor fabricated in 65-nm technology. To achieve high energy efficiency, three optimization techniques are proposed. First, each processing element (PE) supports bit-width adaptive computing to meet various bit-widths of neural layers, which raises computing throughput by 91% and improves energy efficiency by 1.93× on average. Second, PE array supports on-demand array partitioning and reconfiguration for processing different NNs in parallel, which results in 13.7% improvement of PE utilization and improves energy efficiency by 1.11×. Third, a fused data pattern-based multi-bank memory system is designed to exploit data reuse and guarantee parallel data access, which improves computing throughput and energy efficiency by 1.11× and 1.17×, respectively. Measurement results show that this processor achieves 5.09-TOPS/W energy efficiency at most.**

***Index Terms*— Energy efficiency, hybrid neural networks (hybrid-NNs), memory banking, reconfigurable computing, resource partitioning.**

I. Introduction

DEEP neural networks (NNs) have been widely used and achieved extraordinary accuracy in many intelligent applications such as image classification, object detection, speech recognition, action recognition, and scene understanding [1]–[6]. Deep NNs can be categorized into convolution-based NN (ConvNet), fully connected NN (FCNet), and recurrent NN (RNN). ConvNet is fully composed of convolutional layers and pooling layers, which is good at visual feature extraction. FCNet is composed of multiple fully connected layers, which is usually used for classification. RNN is composed of fully connected layers with feedback paths and gating operations, which performs well in sequential data processing. These three kinds of NNs can be used individually. For example, a standalone ConvNet can be used as a fully convolutional network [7] for image segmentation. Meanwhile, these NNs can be combined to construct more complicated NNs. For example, as shown in Fig. 1, AlexNet is a combination of 5-layer ConvNet and 3-layer FCNet, which is proposed for image classification. Long-term recurrent convolution network (LRCN) [6] is a cascade of 5-layer ConvNet, 3-layer FCNet, and 2-layer RNN, which is capable of image captioning. CLDNN [5] consists of 2-layer ConvNet, 6-layer FCNet and 2-layer RNN to achieve speech recognition. We define these kinds of combinations of ConvNet, FCNet, and RNN as hybrid-NNs. With the development of deep NNs, hybrid-NNs exhibit a great advantage on cognitive tasks and are becoming more and more important [8], [9].

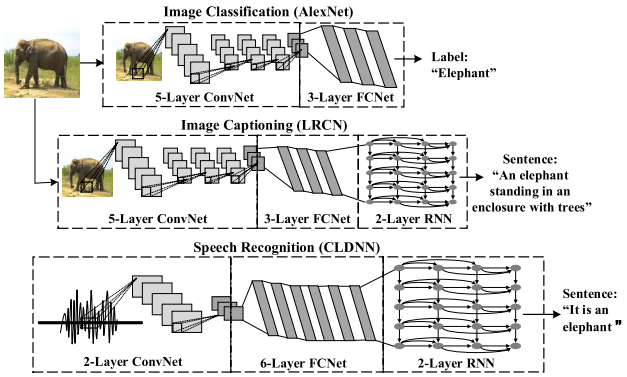


Fig. 1. Three examples of hybrid-NNs: AlexNet [10] for image classification, LRCN [6] for image captioning, and CLDNN for speech recognition [5].

Hybrid-NNs are usually high-throughput algorithms requiring a large number of parallel computations and memory accesses. Taking LRCN as an example, it requires tens of megabytes of weights and billions of operations in a single inference pass, resulting in less than 100-GOPS/W energy efficiency on general GPU or CPU platform [11]. To improve performance and energy efficiency, several NN processors are proposed [12]–[19]. However, most of them only focus on optimizing convolutional computations. This paper presents a hybrid-NN processor (called "Thinker") [20], which is designed to address the three special features of hybrid-NNs.

First, compared with the traditional convolutional NNs, a hybrid-NN has more kinds of non-CONV operations, which

Manuscript received August 4, 2017; revised October 15, 2017; accepted November 20, 2017. Date of publication December 14, 2017; date of current version March 23, 2018. This paper was approved by Guest Editor Makoto Ikeda. This work was supported by the National Natural Science Foundation of China under Grant 61774094. (Shouyi Yin and Peng Ouyang contributed equally to this work.) *(Corresponding author: Shouyi Yin.)*

S. Yin, S. Tang, F. Tu, X. Li, S. Zheng, T. Lu, J. Gu, L. Liu, and S. Wei are with the Institute of Microelectronics, Tsinghua University, Beijing 100084, China (e-mail: yinsy@tsinghua.edu.cn).

P. Ouyang is with the School of Electronics and Information Engineering, Beihang University, Beijing 100083, China.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JSSC.2017.2778281

take up a higher proportion of total computations. Meanwhile, a hybrid-NN has the feature of fault tolerance and resilience, which enables the low precision quantization of some layers with very little accuracy loss [21]. Most of the state-of-the-art designs mainly focus on improving the performance and energy efficiency of convolutional layers [13], [22], [23]. However, if only the convolutional layers in a hybrid-NN are accelerated, the non-CONV layers will be the bottleneck of performance, which was demonstrated in [18] and [24]. Therefore, in [18], DNPU processor is proposed to accelerate both the CONV layers and RNN layers rather than the CONV layers only. However, in DNPU, the hardware resources for CONV layers and RNN layers are independent and not reusable, which limit the flexibility and efficiency to process complex and various hybrid-NNs in different applications. Hence, it is necessary to design reconfigurable computing units which can support various operations with low bit-width to accelerate the entire hybrid-NN.

Second, the time-multiplexing (TM)-based computing flow, which is adopted in the existing NN processors, is inefficient for processing hybrid-NNs. In TM-flow, the neural layers are processed one by one. By transforming fully connected (FC) layers into a unified representation of CONV layers, both the CONV and FC layers can be processed in TM-flow on the existing NN processors [13], [24]. However, in a hybrid-NN, ConvNet, FCNet, and RNN show large difference of operational density. Accelerating them in TM-flow will cause low utilization of computing resources when processing FCNet and RNN, resulting in worse performance. Thus, it is important to design a high-efficient computing flow for hybrid-NN.

Third, the memory storage and access mechanism in existing NN processors, which are specially designed for convolutional computations, are not suitable for FCNet and RNN. Most of the existing NN processors employ spatial architecture for parallel computing to improve throughput and energy efficiency. To enable parallel data access, the multi-bank memory system and corresponding access mechanism are designed according to the specific data flow in convolutional computations [12], [13]. However, the data access pattern in FCNet and RNN is different from that in ConvNet. The multi-bank memory system designed for convolutional data flow cannot guarantee providing all required data in parallel for both CONV and non-CONV layers, which results in low resource utilization. Therefore, it is required to design a flexible and high-efficient multi-bank data management and access mechanism.

Thinker processor mainly has four key innovations.

1) Two types of reconfigurable processing elements (PEs) supporting *bit-width adaptive computing* and covering *all basic operations in hybrid-NN* can efficiently process ConvNet, FCNet, and RNN.
2) Two 16×16 heterogeneous reconfigurable PE arrays supporting *on-demand array partitioning* (ODAP) to process ConvNet, FCNet, and RNN in parallel can efficiently exploit the utilization of computing resources.
3) A multi-bank memory system employing *fused pattern-based memory banking* (FPMB) strategy meets flexible data access demand in hybrid-NNs and exploits data reuse to enhance computing throughput.
4) An automatic mapping flow supporting ODAP and memory banking to map an arbitrary hybrid-NN onto Thinker processor.

II. Hybrid Neural Network Analysis

A hybrid-NN is usually a combination of two or three of ConvNet, FCNet, and RNN. In this section, we take one typical hybrid-NN, LRCN, as an example to analyze the characteristics of hybrid-NN. Table I summaries the parameters and data statistics of LRCN.

1) Various Types of Operations and a Large Proportion of Total Computations for FCNet and RNN: In ConvNet, the main operations are convolutions, in which a 3-D filter is used to extract features from the input data. An output feature map is generated by sliding a 3-D filter on the input feature map. For instance, it receives Ch_{in} input feature maps and outputs Ch_{out} feature maps. The Ch_{out} feature maps correspond to the filtered results of Ch_{out} 3-D filters. Each 3-D filter is operated on Ch_{in} input feature maps to generate one output feature map. Besides, ConvNet also contains the pooling operations and ReLU operations. FCNet mainly performs matrix multiplication. The input data include vectors and weighting matrixes, while the output data are vectors. In FCNet, each FC layer is generally followed by nonlinear layers such as ReLU or Sigmoid, and so on. In RNN, long–short term memory (LSTM)-based network [25] is widely used. In LSTM-based RNN, the operations mainly include fully connected layers and element-wise multiplication that is in charge of gating function. In this paper, we use the term "FCNet/RNN" to indicate fully connected layers in these two networks and "RNN-gating" to indicate the rest RNN operations. In LRCN, FCNet and RNN take up a large proportion (47.7%) of total computations, which means both FCNet and RNN also demand accelerating rather than ConvNet only.

2) Various Operational Density of Different Layers: As shown in Table I, different layers in LRCN are characterized by operational density. The layers in ConvNet are computation-intensive and the layers in FCNet/RNN are memory intensive. ConvNet exhibits high data reusability in which a lot of multiply–accumulate (MAC) operations are taken with small memory footprint for weights. In contrast, FCNet/RNN needs a lot of MAC operations but no weight can be reused. Since a hybrid-NN is the cascade of several CONV, FC, and RNN layers, it exhibits time-varying features. These variations cause large fluctuation in computing resource utilization and memory bandwidth utilization, resulting in resource underutilization and performance degradation. Therefore, the hybrid-NNs demand a new computation flow to balance computation-intensive layers and memory-intensive layers, which can improve overall resource utilization.

3) Low Bit-Width Quantization for Different Layers: Since the hybrid-NNs are inherently fault tolerant, some layers can be quantized into low bit-widths with limited algorithm accuracy loss [21]. To evaluate accuracy loss, LRCN is trained in Caffe framework with different bit-widths of weights. The Microsoft COCO Captions data set [26] is used, which

TABLE I
STATISTICS OF LRCN

Neural network	Layer number	ConvNet:(K,S) FCNet/RNN:(I_{len}, O_{len})	Layers operation	MAC-ops (MOPS)	Weights (MB)	Feature maps (MB)	OP density (MOP/MB)	Bit-width (bits)	Acc loss
ConvNet	1	(11,4)	MAC Pooling ReLU	105.4	0.039	0.58	3011.4	9	1.8%
	2	(5,1)		447.9	0.61	0.37	734.3	8	
	3	(3,1)		149.5	0.88	0.13	196.9	8	
	4	(3,1)		224.3	1.33	0.13	168.6	8	
	5	(3,1)		149.5	1.00	0.09	150.2	9	
FCNet	6	(9216,4096)	MAC ReLU	37.8	37.8	0.008	1.0	8	
	7	(4096,4096)		16.8	16.8	0.008	1.0	8	
	8	(4096,1000)		4.1	4.1	0.002	1.0	8	
RNN	9	(2024,4096)	MAC,Tanh ReLU,Sigmoid Element-wise	8.29× 50 iterations	8.29	0.008	50.0	8	
	10	(2048,4096)		8.39× 50 iterations	8.39	0.008	50.0	8	

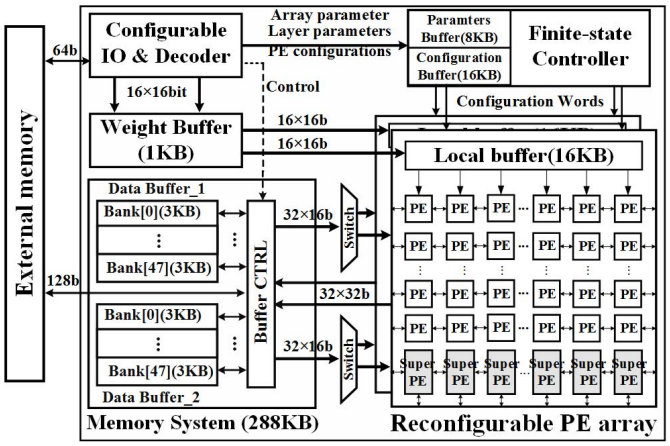


Fig. 2. Architecture overview.

has 82 783 train images and 40 504 validation images. The accuracy is tested on 5000 images randomly chosen from validation set. As shown in Table I, when quantizing LRCN with data width of 9/8/8/8/9/8/8/8/8/8 bit for each layer, the accuracy loss (Acc loss) is only 1.8%, compared to 16-bit quantization for all layers. Thus, the computing logics should support various bit-width operations to fully exploit the computing power of hardware resources.

4) Different Data Access Patterns and Data Reusability in ConvNet and FCNet/RNN: In LRCN, as shown in Table I, various kernel sizes K and strides S exist in ConvNet, and various input–output data lengths (I_{len}, O_{len}) exist in FCNet and RNN, which make the data access pattern of each layer quite different. Moreover, both input feature points and weights can be reused in ConvNet, while a little data and weights can be reused in FCNet and RNN. For ConvNet, the memory system in NN processor should be able to leverage data reuse to reduce redundant memory accesses. For FCNet/RNN, it should be able to provide enough data in parallel to fully exploit computing resources. Therefore, a flexible and configurable on-chip memory system is required for accelerating not only ConvNet but also FCNet and RNN.

III. SYSTEM ARCHITECTURE

A. Overall Architecture

Fig. 2 shows the top-level architecture of the proposed hybrid-NN processor. Two 16×16 heterogeneous PE arrays are the main computing units. Each array can be partitioned into sub-arrays for different functions, and PEs can be configured to execute bit-width adaptive operations. The buffer controller manages two 144-KB multi-bank on-chip buffers to supply data for PE arrays. The overall execution is controlled by the finite-state controller, which loads weights and configuration context through configurable IO and decoder unit. One 1-KB shared weight buffer and two 16-KB local buffers are used to prepare weights for PE arrays.

1) Configurable Heterogeneous PE Arrays: There are two types of PEs in the heterogeneous PE arrays: general PE and super PE. Both of them include a bit-width adaptive computing unit, which can be configured to execute two 8 × 16-bit multiplications or one 16 × 16-bit multiplication. As shown in Fig. 3, general PE supports MAC operations of ConvNet, FCNet, and RNN. Its function is controlled by a 5-bit configuration word ($s_1, s_3, s_6, s_7, s_{11}$). Super PE is the extension of general PE with five more operations: pooling, tanh, sigmoid, scalar multiplication and addition for pooling layers, and RNN-gating operations, under the control of a 12-bit configuration word ($s_0 \sim s_{11}$). The multiplier and adder are reused in almost every function, except for pooling, sigmoid, and tanh in super PE. Therefore, the average unused PE area in different functions is small. Moreover, the power overhead of unused units in each function is saved by clock-gating. Fig. 4 shows the configured datapaths of major functions. Besides, each PE supports zero-skipping to reduce redundant operations and power consumption. Each array has three groups of IO ports for data input–output at the left, right, and bottom edges, respectively. The input feature points are loaded to PEs at the left/right edge, and are horizontally shifted to the PEs inside the array, while the output value on each PE is shifted to the edge of array in the opposite direction. Weight bus on every column is used to broadcast weights to 16 PEs. With these hardware implementations, each PE array can be partitioned into at most four different functional sub-arrays with independent IO ports, which can execute in parallel.

2) On-Chip Memory System: Two 144-KB multi-bank SRAM data buffers (Data Buffer_1 and Data Buffer_2) store intermediate data between NN layers and exploit data reuse. In order to support array partitioning (AP), each data buffer is divided into two sub-buffers for CONV and FC sub-arrays. During the computation of a specific layer, one sub-buffer

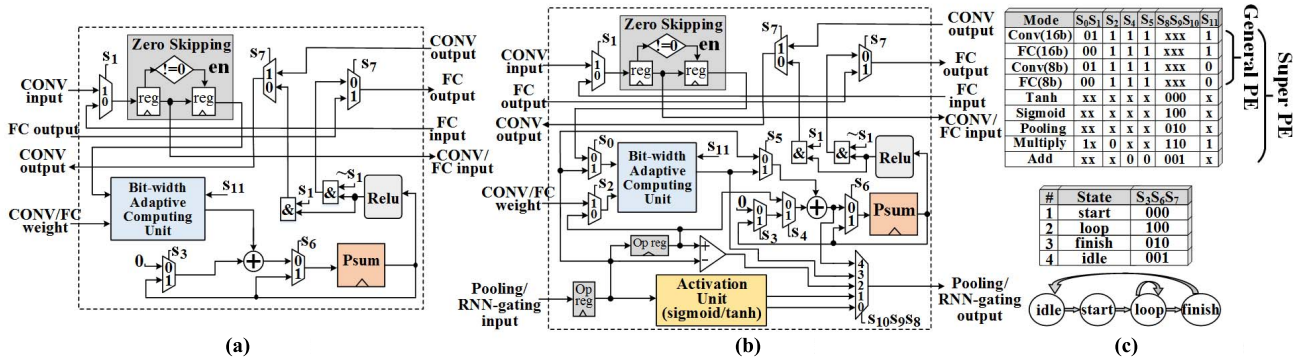


Fig. 3. PE architecture. (a) General PE. (b) Super PE. (c) Config: Mode + State.

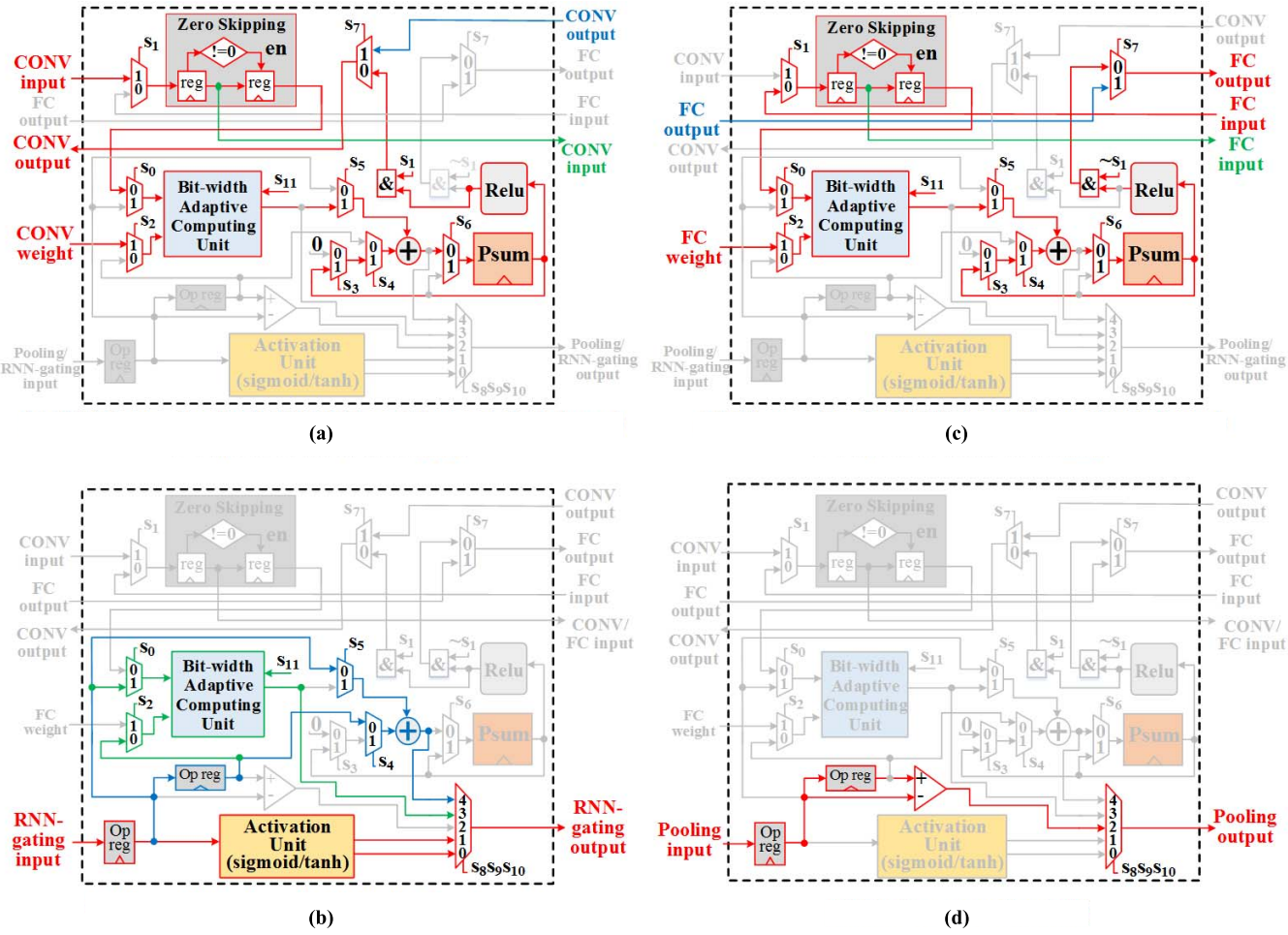


Fig. 4. Configured datapaths of major functions. (a) CONV operation (red)/CONV input forward path (green)/output backward path (blue). (b) FC operation (red)/FC input forward path (green)/output backward path (blue). (c) Sigmoid, Tanh (red)/multiplication (green)/addition (blue) in RNN-gating. (d) Pooling operation (red).

provides input data to the arrays and the other stores the output data from the arrays. For next layer, two buffers exchange their functions. There are 48 banks in each buffer so that at most 48 × 16 bits data can be provided in parallel to PE arrays. Each memory bank works in double buffering way. By this mechanism, the data exchange between PE arrays and on-chip buffers is concurrent with the data prefetching between on-chip buffers and off-chip DRAM. Buffer controller manages the data access and data organization in on-chip buffers.

The 1-KB weight buffer is used to store the weights loaded from external memory and provides weights to PE arrays. In each PE array, a 16-KB local buffer exploits weight reuse, which provides 16 weights for 16 PE columns in parallel.

3) Finite-State Controller: The finite-state controller is used to configure Thinker processor at three level: PE array level, neural layer level, and PE level, as shown in Fig. 5. First, the array parameter is a 66-bit word, which indicates the PE AP parameters, batch size (BS), the number of NN layers,

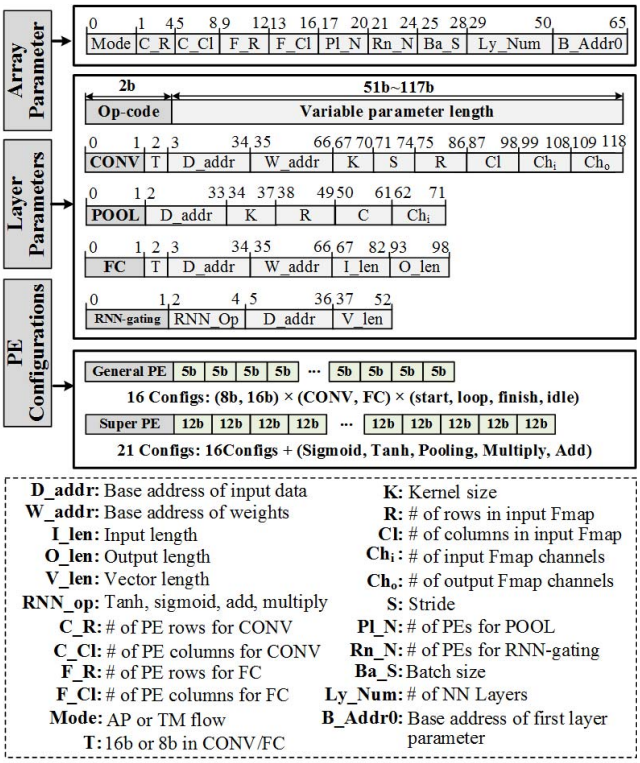


Fig. 5. Three-level configuration hierarchy: array parameters, layer parameters, and PE configurations.

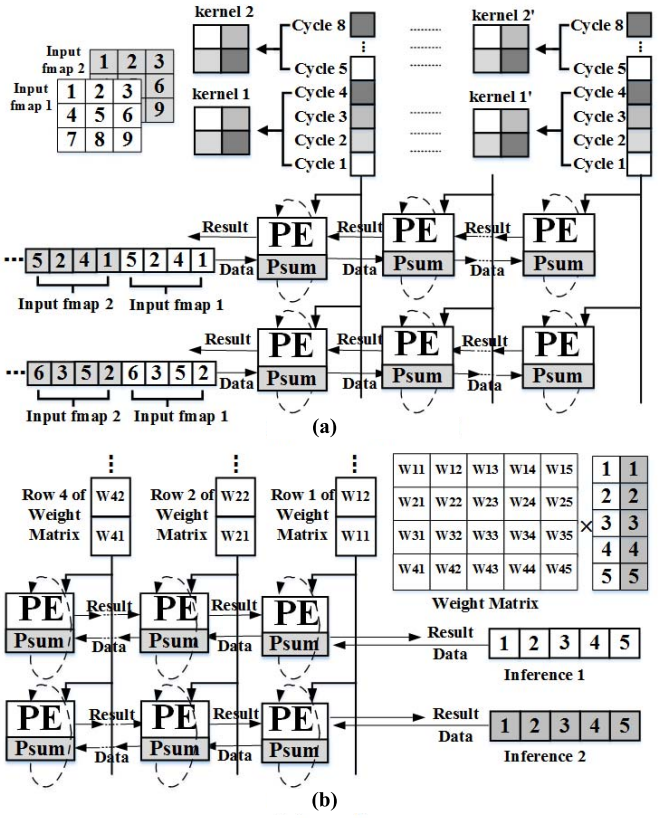


Fig. 6. Computing flow of (a) CONV operation and (b) FC operation.

and the base address of layer parameters. Second, the layer parameter words are used to control the processing of one specific neural layer. It contains memory address of input data and weights, as well as kernel size, the number of output feature channels, etc. Third, PE configuration words directly control the switches in each PE, thereby determining its function. Sixteen 5-bit words and twenty-one 12-bit words are used to control functions in general PEs and super PEs, respectively.

During runtime, the finite-state controller first reads the array parameters and layer parameters in parameters buffer. Then the controller identifies each PE's function by decoding these parameters and chooses the corresponding configuration words in configuration buffer. Finally, the configuration word for each PE is sent via the point-to-point connections between PE arrays and configuration buffer, which trade 7% area overhead for configuring two arrays in one cycle.

4) IO and Decoder: The weights and configurations are loaded through a configurable IO. The weights are compressed by two-symbol Huffman coding to exploit the sparsity and decoded by the decoder unit. To support the flexible length of layer parameter words shown in Fig. 5, this decoder first reads the highest two bits, which indicate the layer type, then loads the corresponding length of word according to it. The input and output data are transferred directly between on-chip buffers and external memory. To fit the buffer capacity, input data are tiled.

B. Computation Flow in PE Array

In PE array, the computation of output points are fixed on the respective PEs, reusing weights, and input feature points in the vertical and horizontal directions, respectively (also known as output stationary dataflow).

For convolutional operations, as shown in Fig. 6, input data are loaded from on-chip buffers into the leftmost PE column via left IO ports. In each cycle, these data are shifted one step to right neighboring PEs via forward path. PEs in one column compute different points of the same output feature map in parallel, and different columns of PEs are assigned with respective output feature maps. When an output point is completed, the result is shifted from right to left through the backward path, and is finally sent to the buffer via left IO ports. In this way, there is no need to add extra IO ports inside the array. The extra cycle overhead of shifting in the accumulation of a 3-D kernel is at most 30 cycles, with 15-cycle input delay and 15-cycle output delay for the right-most column of PEs. As it takes commonly thousands of cycles to finish a 3-D kernel, this overhead is negligible. For example, it accounts for only 2.2% of total AlexNet computing cycles.

For fully connected operations, input data are loaded to the array via right IO ports and moved one step to PEs at left side in each cycle. The output points are shifted from left to right. PEs in the same row reuse the input data to compute multiple points in the same output layer, while different rows are in charge of respective inference passes in one batch, reusing the weights of fully connected layers.

Before being loaded to the array, input data of CONV layers are tiled to fit the size of on-chip data buffers. Assuming that $m \cdot n$ PEs are allocated for convolution, the size of kernel is k, the number of input channels is Ch, and stride is 1

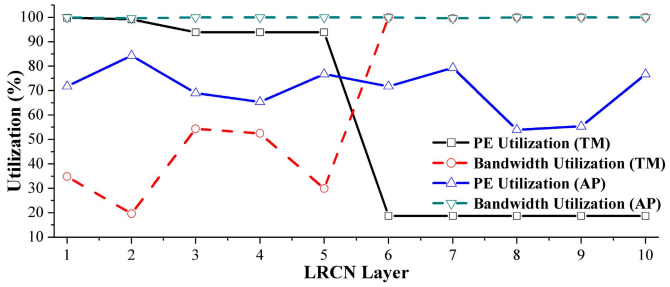


Fig. 7. Utilization of PEs and bandwidth, evaluated on each layer of LRCN.

(for concision). Therefore, $(m + k - 1) \cdot k \cdot \text{Ch}$ input points are needed to compute $m \cdot n$ output points, and are adopted as the tile size for this layer. The input size of FCNet/RNN is relatively small, e.g., 8 KB for 4096 points of a typical FC layer. They can be loaded entirely into the on-chip data buffers with no tiling.

Pooling and RNN-gating operations are carried out on the super PEs at the bottom row of the array, which read input data from the on-chip buffers via the bottom IO ports, and send the results back.

To map a hybrid-NN on PE arrays, there exists a straightforward TM-flow, which schedules all neural layers sequentially. However, in this layer-by-layer manner, the computation-intensive ConvNet will not make full use of the DRAM bandwidth, while huge DRAM accesses of FCNet and RNN will result in the underutilization of PEs. We take LRCN in TM-flow as an example to analyze DRAM bandwidth and PE utilization during runtime. As shown in Fig. 7, PE utilization rate achieved by TM-flow is only 18.75% during the execution phase of FCNet/RNN, while its lowest bandwidth utilization rate is only 19.63% during the execution phase of CONV layers. On average, PE utilization rate and memory bandwidth are only 75.68% and 76.96%, respectively.

It is implied that the complementary need for computing and memory accessing can be combined to make the maximum use of resources. Therefore, an AP-flow is proposed in this paper, in which majority of PEs are allocated to computation-intensive ConvNet and most of the bandwidth are allocated to memory-intensive FCNet/RNN. Thinker's multi-bank on-chip buffers, distributed IO ports, and independent computation flows of different operations enable the PE array to be flexibly partitioned into four sub-arrays to process CONV, FC, pooling, and RNN-gating operations, respectively. Fig. 8(a) illustrates processing LRCN in AP-flow. CONV array is assigned with 15×13 general PEs; 15×3 general PEs are assigned for FC array. Nine and seven super PEs are assigned for pooling and RNN-gating, respectively. Each on-chip buffer is partitioned into a 33-bank CONV buffer and a 15-bank FC buffer, and the BS is set as 15. Fully connected operations in RNN share the same FC array with FCNet and the rest RNN-gating operations are executed on super PEs. In CONV array, different output channels are mapped onto separate columns, obtaining 13 times of input data reuse and 15 times of weights reuse. In FC array, multiple inference passes are computed in parallel, the reuse times of input data and weights

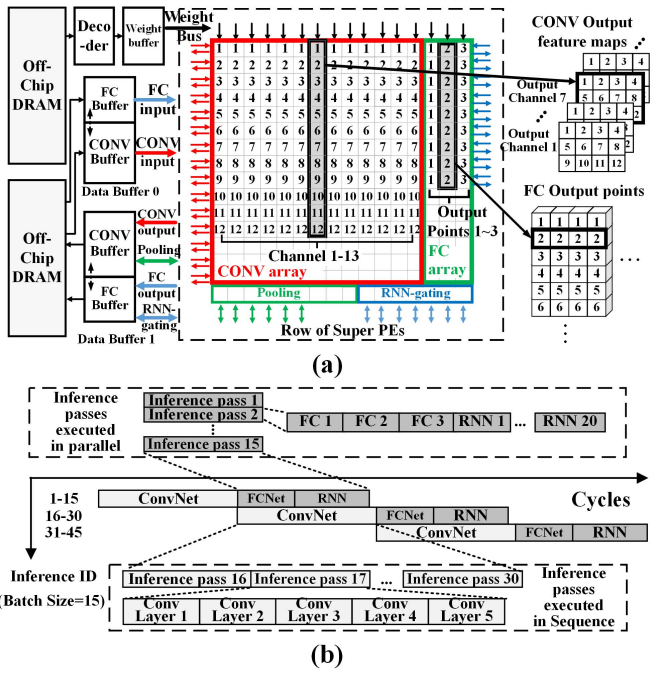


Fig. 8. (a) AP and AP-flow for LRCN. (b) Batch level pipelining.

are 3 and 15 (BS), respectively. As illustrated in Fig. 8(b), in AP-flow, the executions of CONV and FC layers in one batch are scheduled in two consecutive pipeline stages. Before FC layers start, their input data are already prepared in the previous pipeline stage. ConvNets of all inference passes in one batch are scheduled sequentially in one pipeline stage, while FCNets/RNNs of different inference passes in one batch are executed concurrently on separate rows of FC array, in order to exploit weights reuse. With AP-flow, we improve overall PE utilization by 10.3%, and bandwidth utilization by 21.02%.

The optimal partitioning parameters, such as the size of CONV/FC array and the number of banks in CONV/FC buffer, vary with different hybrid-NN topologies. In some extreme cases, the computations and memory accesses of ConvNet are absolute majority in a hybrid-NN, and the TM-flow may even outperform AP-flow. Therefore, we propose an ODAP algorithm in Section IV.

IV. Optimization Techniques

In this section, we propose three optimization techniques, which mainly correspond to the three features of hybrid-NNs.

A. Bit-Width Adaptive Computing

Bit-width scaling is an effective approach to improve efficiency in NN processors [12], [18], [27]. In [18], an look-up table-based reconfigurable multiplier is proposed to support 4-/8-/16-bit multiplication. In [27], two 8-bit multiplications can be performed in parallel. In [12], a subword-parallel MAC is proposed. In Thinker processor, the bit-width adaptive computing unit is designed based on digital logics and has higher efficiency and resource utilization for 8-/16-bit operation.

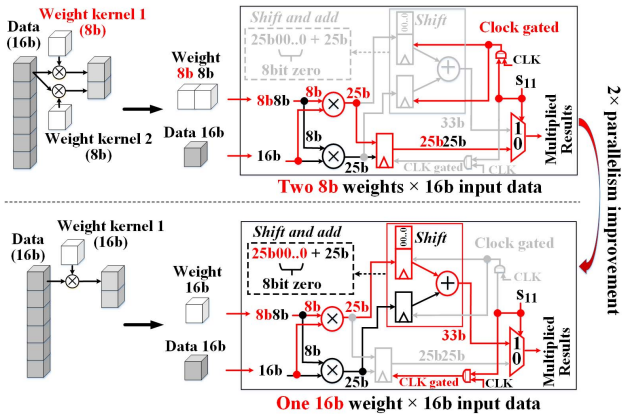


Fig. 9. Bit-width adaptive computing.

Each PE has two 8 × 16-bit multipliers, supporting two computing mode. If S_{11} in PE configuration word is 0, two 8 × 16-bit multiplications are processed in parallel. Otherwise, two multipliers are combined as one 16 × 16-bit multiplier to support 16-bit operation. In each mode, the unused datapaths are clock-gated to reduce power consumption.

As shown in Fig. 9 (top), when the bit-width of weight is less than 8, two weights from different ConvNet kernels or FCNet/RNN neurons are concatenated as one 16-bit weight. The high 8-bit part and low 8-bit part are multiplied with the same 16-bit input data on two multipliers, respectively. The result from each multiplier is represented as one 25-bit word including one reserved bit for carry. Then two 25-bit results are concatenated as a 50-bit word, which is sent out by setting S_{11} as 0. In this mode, since two operations are performed on two separate multipliers, no guard band is required in the input data. Compared to our design, the INT8 optimization in [27] requires at least 9-bit guard band to separate the results of two 8-bit multiplications. Therefore, our design achieves higher efficiency when performing 8-bit multiplications.

When the bit-width of weight is larger than 8-bit, these two multipliers are combined as one 16 × 16-bit multiplier. As shown in Fig. 9 (bottom), the results from two multipliers are sent to the *shift and add* unit (shown in the red-line box). The result of high 25-bit part is shifted by 8-bit with zero filling. Then it adds with the low 25-bit part. By setting S_{11} as 1, the result is output. Compared to [12], the multipliers in our design are all utilized in either 8-bit mode or 16-bit mode. Therefore, our design has higher resource utilization.

B. Fused Data Pattern-Based Memory Banking

In AP-flow, PE array is partitioned into several independent sub-arrays, including CONV array and FC array, to execute ConvNet and FCNet/RNN concurrently. Input feature points for both CONV array and FC array need to be loaded from multi-bank on-chip buffer in parallel. Since ConvNet and FCNet/RNN have quite different data access patterns, in order to ensure the parallel data access, the multiple memory banks are partitioned to two groups, CONV buffer and FC buffer,

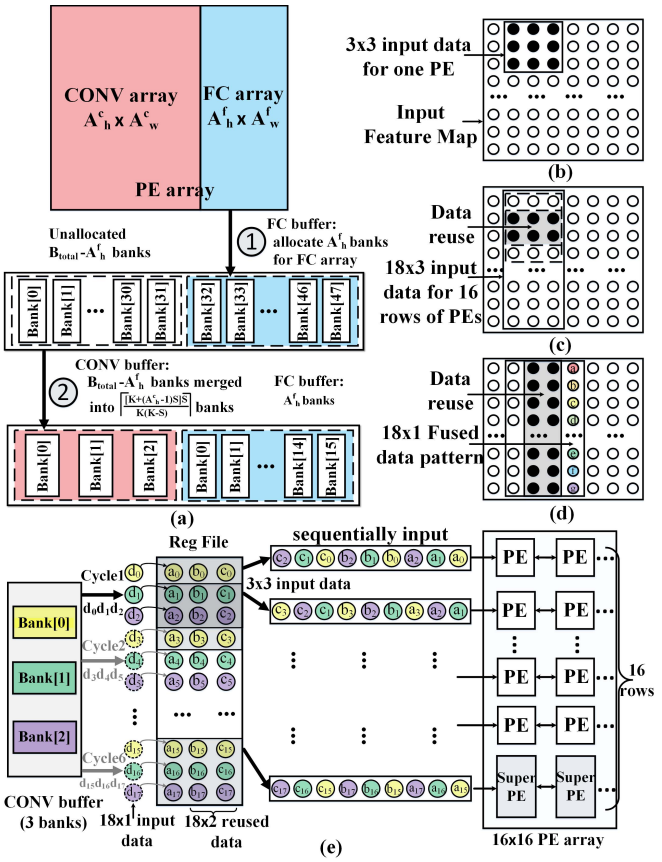


Fig. 10. Fused data pattern-based memory banking. (a) Bank allocation for CONV array and FC array. (b) 3x3 input data for one PE. (c) 18x3 input data for 16 rows of PEs. (d) 18x1 input data in a fused data pattern. (e) Example of data flow.

TABLE II
NOTATION ILLUSTRATION

Input: hybrid NN parameters	
DL, WL	bit-width of activations and weights
$\theta \in \{1, 2\}$	number of concurrent MACs executed on one PE
FC_{in}, FC_{out}	number of input/output nodes in FCNet/RNN layer
Ch_{in}, Ch_{out}	number of input/output channels in ConvNet layer
R, C; H, W; K; S	height, width of input feature map, output feature map; kernel size; stride
Input: chip specifications	
f; $BW_{\{a,c,f\}}$	frequency; bandwidth allocated for data; effective average bandwidth for ConvNet and FCNet
$B_{\{total,m\}}$	number of total banks; minimum number of banks required for ConvNet
Output: performance results	
$CC_{\{conv,ConvNet\}}$ $CC_{\{fc,FCNet/RNN\}}$ $CC_{\{trans\}}$ $CC_{\{AP,TM\}}$	computation cycles of one layer and the entire of ConvNet, FCNet/RNN; data transmission cycles of one layer; total computation cycles of AP-flow and TM-flow
Output: partitioning parameters	
A_h, A_w, A_h^c, A_h^f, A_w^c, A_w^f	number of total rows, columns; rows allocated for ConvNet and FCNet; columns allocated for ConvNet and FCNet
BS	number of inference passes in a batch

which provide input data for CONV array and FC array with corresponding access pattern, respectively.

Assume that the size of CONV array and FC array is $A_h^c \times A_w^c$ and $A_h^f \times A_w^f$, respectively, as shown in Fig. 10(a). A_h^c and A_h^f indicate the number of PE rows. The other notations are explained in Table II. In FC array, there are A_h^f

rows of PEs executing concurrently and each row calculates the output maps of different inference passes. Since there is no data reuse between any adjacent PE rows, A_h^f input points in total are required to be loaded in parallel. Therefore, A_h^f banks are assigned to FC buffer. The rest $B_{\text{total}} - A_h^f$ banks can be used for CONV array.

In CONV array, each PE is in charge of the calculation for one output feature point. For a kernel of size K, since the output stationary dataflow is adopted, each row of PE array requires a set of $K \times K$ input feature points to calculate one output feature point. When different rows of PEs execute concurrently, input points for adjacent rows are overlapped. One union set is used to represent all the required input points for the A_h^c rows of PEs considering data reuse. The height and width of this union set are $H_u = K + (A_h^c - 1) \cdot S$ and $W_u = K$. An example is given in Fig. 10(b)–(d), where a 3×3 ($K = 3$) kernel-based convolution is performed with stride S of 1 in a 16-row CONV array. In Fig. 10(c), $H_u = 3 + (16 - 1) \cdot 1 = 18$ and $W_u = 3$, so a set of 18×3 input points are required for 16 PE rows. With the kernels slide across the input feature map horizontally, the corresponding union sets are overlapped, as shown in Fig. 10(d). Considering data reuse, $H_u \times (K - S)$ input points will be reused and retained for the next round of calculations. Thus, only $H_u \times S$ input points need to be loaded for each round, which is termed as a fused data pattern $\vec{P}$. In Fig. 10(d), a fused data pattern consists of 18×1 input data. In this example, two-thirds of redundant memory accesses can be reduced.

To avoid execution stall of PE array, all the points in a fused pattern must be loaded within the time span of consuming the reused $H_u \times (K - S)$ points. With the output stationary data flow, PE array takes K clock cycles to consume one column of points in the input feature map. So all input points in a fused data pattern should be loaded within $K \cdot (K - S)$ clock cycles. Given the data amount of a fused data pattern and the corresponding clock cycles of loading data, the minimum number of loaded data per clock cycle is

$$B_m = \left\lceil \frac{[K + (A_h^c - 1) \cdot S] \cdot S}{K \cdot (K - S)} \right\rceil \tag{1}$$

which is also the minimum number of banks required for CONV buffer. If there exist redundant banks, the remaining $B_{\text{total}} - A_h^f$ banks are merged into B_m virtual banks.

In CONV layers, each input feature point can be represented as $\vec{p} = I[r][c][ch_{\text{in}}]$, where $0 \le r \le R-1$, $0 \le c \le C-1$, and $0 \le ch_{\text{in}} \le \text{Ch}_{\text{in}}$. In order to enable the parallel access of fused data pattern $\vec{P}$, each point in $\vec{P}$ needs to be mapped to a unique bank. Therefore, the input feature point $\vec{p}$ should be mapped to the virtual bank number $B(\vec{p}) = r \bmod B_m$. In ConvNet, the output map of each CONV layer is written back to multi-bank on-chip buffer according to the same rule and serves as an input map of the next layer.

In order to support data reuse, a register file is used to store $H_u \times K$ input points. The buffer controller dispatches the input points from register file to CONV array. Each input feature point for $K \times K$ filter is sequentially input into the corresponding row of PE array. As shown in Fig. 10(e), the minimum bank number $B_m = \lceil (18 \cdot 1/3 \cdot 2) \rceil = 3$. Thus, 18×1 input data need to be fetched from a 3-bank CONV buffer within 6 clock cycles. Then the input points will be stored in the register file, so that the previously loaded 18×2 input points can be reused to form the complete 18×3 input points. Since 16 rows of PEs execute concurrently, 18×3 points will serve as $16 \times 3 \times 3$ input points considering data reuse. Each 3×3 input points will be sequentially fed into one row of PEs via left IO port.

C. On-Demand Array Partitioning

The performance of AP-flow is determined by the partitioning results of PE array. Given a specific hybrid-NN, there exist an optimal set of variables (A_h^c, A_w^c, A_h^f, A_w^f, BS), leading to the highest PE utilization and the best performance. Algorithm 1 is proposed to find the optimal partitioning parameters.

Algorithm 1 Finding Optimal PE Array and Memory Banks Partitioning

```
//In_c, Out_c, Weight_c, Weight_f represent DRAM access of each layer.
Initiate :CC_AP = ∞
for BS = 1; BS ≤ A_h − 1; BS++ do
    for A_w^c = 1; A_w^c ≤ A_w − 1; A_w^c++ do
        for A_h^c = 1; A_h^c ≤ A_h − 1; A_h^c++ do
            Initiate :CC_conv, CC_fc, CC_trans, CC_ConvNet, CC_FCNet/RNN, CC_temp, B_m;
            for j = 1; j ≤ CONV_layer_num; j++ do
                B_m[j] = ⌈[K_j+(A_h^c−1)·S_j]·S_j / K_j·(K_j−S_j)⌉
            B_m = max_j{B_m[j]}
            if B_m + BS > B_total then
                continue
            for j = 1; j ≤ CONV_layer_num; j++ do
                CC_conv[j] = ⌈H_j·W_j / A_h^c⌉ · ⌈Ch_out,j / θ·A_w^c⌉ · K_j^2 · Ch_in,j · BS
                CC_trans[j] = max(In_c,j + Out_c,j / BW_a/f, Weight_c,j / BW_c/f)
            CC_ConvNet = Σ_j max(CC_trans, CC_conv)
            for j = 1; j ≤ FC_layer_num; j++ do
                CC_fc[j] = ⌈BS / A_h − 1⌉ · ⌈FC_out,j / θ·(A_h − A_h^c)⌉ · FC_in,j
                CC_trans[j] = Weight_f,j / BW_f/f
            CC_FCNet/RNN = Σ_j max(CC_trans, CC_fc)
            CC_temp = max(CC_ConvNet, CC_FCNet/RNN)
            if CC_AP > CC_temp then
                update CC_AP
```

The essential of AP-flow is to construct a pipeline of inference passes. As shown in Fig. 8(b), AP-flow schedules one batch of hybrid-NN on two consecutive pipeline stages, overlapping ConvNet of current batch with FCNet/RNN of the previous batch. Therefore, we separately calculate the computation cycles of ConvNet and FCNet/RNN, and choose

the slower one as initiation interval (II) of pipeline. Our target is to minimize II.

In Algorithm 1, we first check whether variables (BS, A_h^c) of current iteration can meet the requirement of the proposed memory banking technique. If not, we will try the next set of variables (lines 4–5).

Then, the number of total computation cycles of ConvNet is calculated (lines 6–8). In according to AP-flow, $\lceil (H \cdot W / A_h^c) \rceil$ indicates that $H \cdot W$ points on each output feature map are tiled and allocated on A_h^c PE rows. As each PE can process θ MACs simultaneously, $\theta \cdot A_w^c$ output channels are also processed in parallel. $K^2 \cdot \mathrm{Ch_{in}}$ is the number of cycles consumed to compute one output point. Dividing data transmission amount by data transferred per cycle (BW/f) gives the data transmission time, which will become the bottleneck in memory-intensive layers. The final number of ConvNet execution cycles is the sum of the larger one of $\mathrm{CC_{conv}}$ and $\mathrm{CC_{trans}}$ in each ConvNet layer (line 8).

Next, we calculate the number of execution cycles of FCNet/RNN in a similar way (lines 9–11). Each inference pass in one batch is executed on one specific row, and output points distributed on $(A_h - A_h^c)$ columns. FC_i equals the number of cycles to calculate one output point. $\mathrm{CC_{trans}}$ of FCNet/RNN is dominated by transmission cycles of weights.

The processing time of pooling and RNN-gating can be hidden in ConvNet or FCNet/RNN computation cycles. The number of super PEs allocated for these two functions is in proportion to the number of their operations. Finally, the computation cycles of one batch is determined by slower sub-array (line 12).

Similarly, the performance of TM-flow can be obtained by assigning full array to each layer, utilizing the same equations, and summing up the ConvNet and FCNet/RNN computation cycles rather than selecting the bigger one: $\mathrm{CC_{TM}} = \mathrm{CC_{ConvNet}} + \mathrm{CC_{FCNet/RNN}}$. Therefore, given a specific hybrid-NN, we can find the best computation flow (TM or AP) and the corresponding AP parameters.

D. Compilation Flow and Configuration Process

The integrated compilation flow is illustrated in Fig. 11. With hybrid-NN parameters and chip specification imported, the partitioning model first enters the traversal loop (step 1). In each iteration, a set of feasible variables (A_h^c, A_w^c, A_h^f, A_w^f, BS) are fixed, then the proposed memory banking technique is used to generate optimized buffer partitioning results (step 2). Next, CC_{AP} under given loop variables can be calculated by Algorithm 1 (step 3). After traversing, the performance of TM-flow and AP-flow are compared to determine the final mapping strategy (step 4). Finally, the array and layer parameters are generated from AP results and network topologies, respectively.

Thinker processor usually works as a coprocessor of primary CPU. Before the execution of a hybrid-NN, CPU loads the array and layer parameters into the Thinker's parameter buffer, and PE configuration words into configuration buffer. During runtime, CPU sends the input data and weights to Thinker's data buffers. The configuration process of Thinker

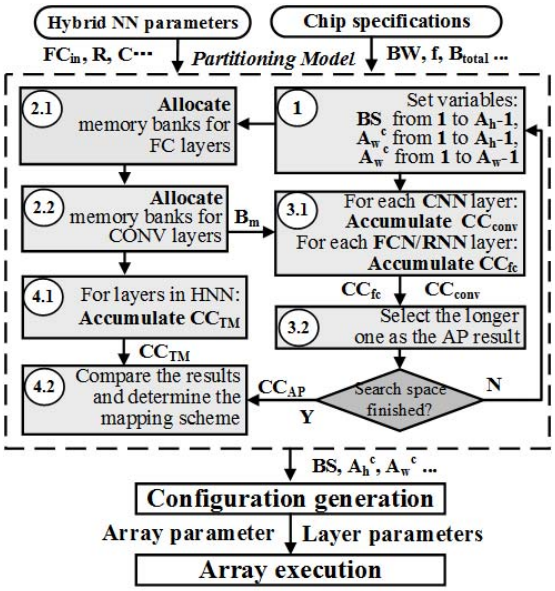


Fig. 11. Compilation flow for mapping hybrid-NN to Thinker processor.

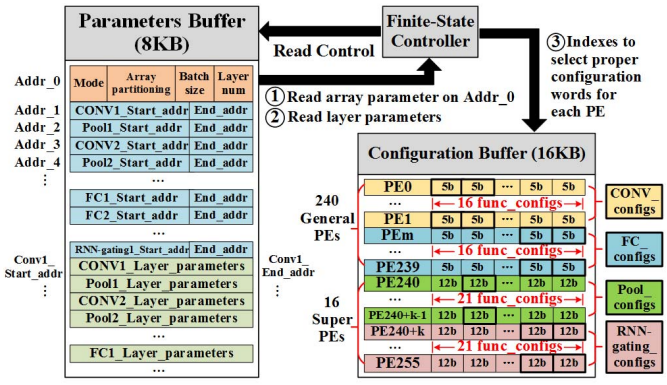


Fig. 12. Runtime configuration process.

is illustrated in Fig. 12. The finite-state controller first reads the array parameter word stored at Address 0 of parameter buffer, which indicates the work mode (AP-flow or TM-flow), partitioning parameters, BS, and the number of neural layers. Then, the following layer parameter words are read. According to these parameter words, the controller identifies the function of each PE and generates the index of configuration word in configuration buffer for each PE. Finally, the corresponding configuration word for each PE is chosen from totally 16 words (general PEs) or 21 words (super PEs) stored in configuration buffer, and sent to the PE array in one cycle.

V. Chip Implementation and Evaluation

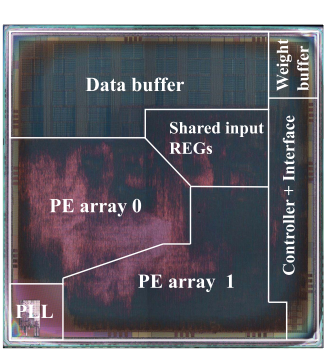
Thinker processor is fabricated in 65-nm LP CMOS technology. Fig. 13 shows the die photograph and chip specification. In the evaluations, AlexNet, CLDNN, LRCN, VGG-LSTM [6] (16-layer ConvNet, 3-layer FCNet, and 2-layer RNN), and Yolo [28] (24-layer ConvNet and 2-layer FCNet) are selected as benchmarks. All the above NNs are trained using fixed point in Caffe framework [29]. In Section V-A, the three

TABLE III
PERFORMANCE COMPARISON BETWEEN BIT-WIDTH ADAPTIVE COMPUTING AND FIXED 16-BIT DESIGN

Benchmarks	BS	Bit-width (bits)		Throughput (GOPS)		Power (mW)		Energy efficiency (TOPS/W)		Acc loss
		Adaptive	Origin	Adaptive	Origin	Adaptive	Origin	Adaptive	Origin	
AlexNet	16	9/8/8/6/6/8/8/8	all 16	302	159	315	320	0.96	0.50	<1.8%
CLDNN	16	all 8	all 16	266	138	314	316	0.85	0.44	<2.0%
LRCN	16	9/8/8/8/9/8/8/8/8/8	all 16	282	155	317	320	0.89	0.48	<1.8%
VGG+LSTM	12	ConvNet:8, FCNet&RNN:12	all 16	327	170	322	325	1.02	0.52	<2.0%
Yolo	16	ConvNet:8, FCNet&RNN:12	all 16	376	192	330	329	1.13	0.58	<3.0%
Average				310.6	162.8	319.6	322	0.97	0.5	<2.5%

TABLE IV
PERFORMANCE COMPARISON BETWEEN AP-FLOW AND TM-FLOW

Benchmarks	Array Utilizations								Array Partitioning in AP-flow	Throughput (GOPS)		Power (mW)		Energy efficiency (TOPS/W)	
	AP				TM					AP	TM	AP	TM	AP	TM
	CONV	FC	Overall	BS	CONV	FC	Overall	BS							
AlexNet	92.4%	44.1%	89.4%	15	96.93%	10.78%	77.65%	16	([15×15],[15×1],[16],[0])	183	159	337	320	0.54	0.50
CLDNN	93.2%	82.1%	91.8%	15	99.4%	18.7%	67.1%	16	([15×14],[15×2],[0],[16])	189	138	344	316	0.55	0.44
LRCN	83.17%	97.91%	85.93%	15	96.48%	42.67%	75.68%	16	([15×13],[15×3],[9],[7])	176	155	331	320	0.53	0.48
VGG+LSTM	92.6%	53.7%	90.1%	15	97.3%	15.0%	81.0%	12	([15×15],[15×1],[15],[1])	189	170	338	325	0.56	0.52
Yolo	82.4%	0.14%	86.6%	1	93.8%	12.6%	93.7%	16	([15×15],[15×1],[16],[0])	178	192	334	329	0.53	0.58
Average (except Yolo)	85.8%	64.8%	84.8%	–	97.5%	17.8%	71.1%	–	–	184	156	337	320	0.55	0.5



Technology	TSMC 65 LP
Chip size	4.4 mm x 4.4 mm
Core size	3.8 mm x3.8 mm
Gates	2.95M(NAND2)
SRAM	348KB
Supply voltage	Core: 0.67~1.2V, IO:1.8V
Frequency	10~200MHz
Peak performance	409.6 GOPS
Operation mode	8/16bit
Convolution	Kernel width:1-16 Kernel height:1-16 Stride: 1-15
Full connection	Input length: 1~65536 Output length: 1~65536
RNN supported	LSTM,GRU,etc.
Other operations	Pooling, Sigmoid, RELU,etc.

Fig. 13. Chip photograph and specification. A live demo available in [30].

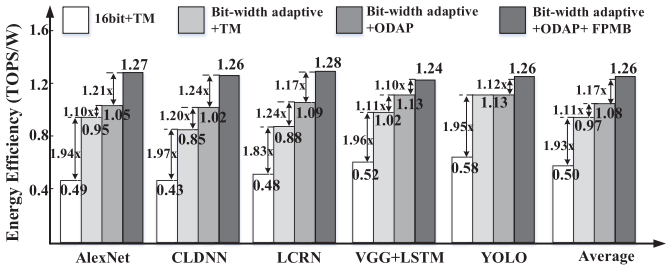


Fig. 14. Combined performance of the proposed techniques.

proposed optimization techniques are evaluated separately on benchmarks. Then the combined effects of the three techniques and the energy efficiency on voltage scaling are evaluated. Section V-B shows the analysis on PE area efficiency and array scalability. Section V-C shows the comparison with state-of-the-art designs.

A. Performance Evaluation

In this section, we first configure Thinker processor in full-precision mode (16 bit) without FPMB and execute all benchmarks in TM mode. We use the obtained results as baseline. Then we adopt each proposed optimization technique individually to evaluate it. Finally, we evaluate the combined effects of all the proposed techniques. The results in Tables III–VI and Fig. 14 are measured at 200 MHz with 1.2-V supply voltage.

1) Analysis on Bit-Width Adaptive Computing: As shown in Table III, the weight of NN layers in all benchmarks can be quantized with low bit-width, resulting in a small accuracy loss (2.5% on average). Two low bit-width operations can be performed in parallel on our bit-width adaptive PE, which exploits the computing power and enhances the throughput. Taking LRCN as an example, the bit-width of weights in each layer is 9/8/8/8/9/8/8/8/8/8, respectively. When LRCN is processed in 16-bit mode, the actual throughput is only 155 GOPS. Since the number of 8-bit operations takes up more than 95% of total computations, the throughput is improved to 282 GOPS when LRCN is processed in bit-width adaptive mode. In general, as shown in Table III, due to the enhanced throughput of PE with comparable power consumption, this technique improves energy efficiency by 1.94× on average over the fixed 16-bit mode in baseline. The optimal BS of each benchmark is also listed.

2) Analysis on On-Demand Array Partitioning: As shown in Table IV, we can see that AP-flow achieves better array utilization than TM-flow for all benchmarks except Yolo.

TABLE V
PERFORMANCE ANALYSIS ON FPMB

Benchmarks	Input size (16b)	BS	Throughput (GOPS)		Power (mW)		Energy efficiency (TOPS/W)		Buffer reads (MB per inference pass)		Buffer writes (MB per inference pass)	
			FPMB	w/o FPMB	FPMB	w/o FPMB	FPMB	w/o FPMB	FPMB	w/o FPMB	FPMB	w/o FPMB
AlexNet	227×227×3	16	189	159	315	320	0.60	0.50	3.38	35.2	1.3	1.3
CLDNN	80×40×1	16	158	138	300	316	0.53	0.44	4.58	44.3	1.9	1.9
LRCN	227×227×3	16	183	155	314	320	0.58	0.48	4.27	39.7	1.3	1.3
VGG+LSTM	224×224×3	12	179	170	306	325	0.58	0.52	70.4	581.4	29.7	29.7
Yolo	448×448×3	16	197	192	320	329	0.62	0.58	104.3	598.2	27.6	27.6
Average	–	–	181	163	335	322	0.58	0.50	37.4	259.8	12.4	12.4

TABLE VI
OVERALL PERFORMANCE WHEN COMBINING THE PROPOSED TECHNIQUES

Benchmarks	BS	Throughput (GOPS)	Power (mW)	DRAM access (for one inference pass)				Energy efficiency (TOPS/W)
				Configurations (KB)	Weights (MB)	Input data (MB)	Output data (MB)	
AlexNet	15	368	290	2.94	13.7	2	1.2	1.27
CLDNN	15	378	298	3.39	21.6	3.6	1.9	1.26
LRCN	15	381	297	3.42	15.2	2.2	1.2	1.28
VGG+LSTM	15	371	298	3.58	245.7	58.1	28.1	1.24
Yolo	1	355	280	3.21	284.1	83.1	25.8	1.26
Average	–	370.6	292.6	3.31	116.1	29.8	11.6	1.26

Correspondingly, high utilization of array makes the throughput of AP-flow 1.18× higher than that of TM-flow, but also leads to 5% higher power consumption. The optimal BSs in AP-flow are different to those in TM-flow. The energy efficiency is improved by 8% ∼ 25%. Taking LRCN as an example, AP-flow allocates 15×13 PEs for CONV operations, 15 × 3 PEs for FC operations, 9 super PEs for pooling, and 7 super PEs for RNN-gating, which denotes as ([15 × 13], [15 × 3], [9], [7]). By sharing memory bandwidth between CONV layers and FC layers, the FC array utilization is 97.91%. But if TM-flow is adopted, the array utilization in the execution phase of FCNet is only 42.67%, since FC layers require high bandwidth. Hence, on average, AP-flow shows 10.24% higher array utilization than TM-flow. For benchmark Yolo, FCNet takes up only 0.01% of total operations, which leads to 0.14% PE utilization of FC array in AP-flow. While in TM-flow, since all PEs are allocated to process CONV operations first, the fast execution of ConvNet compensates the low array utilization in the execution of FCNet, which makes TM-flow achieve better performance than AP-flow. This result is consistent with our previous analysis. Since our compilation flow searches both AP-flow and TM-flow, the best computation flow is guaranteed to be found for this kind of CONV-dominated hybrid-NN.

3) Analysis on Fused Pattern-Based Memory Banking: The proposed memory banking technique maps the input feature points into multiple memory banks according to the fused data pattern and reuses the overlapped data. In contrast, if FPMB is not adopted (denoted as without FPMB), the input feature points are allocated to multiple banks in a rotational manner and each PE reads data without considering overlaps. As shown in Table V, the on-chip buffer accesses are greatly reduced by FPMB. Moreover, the dedicated data mapping method guarantees parallel data access, which eliminates the execution stalls of PE array. Therefore, the throughput is improved by 1.11× with 4% increase of power consumption on average, compared to the case of without FPMB. The final average energy efficiency is improved by 1.16×.

4) Combined Effects of Proposed Techniques on Benchmarks: As shown in Fig. 14, the three proposed techniques can be arbitrarily combined to improve energy efficiency. The baseline (16-bit+TM) energy efficiency of five benchmarks is among 0.43 ∼ 0.58 TOPS/W. By adopting bit-width adaptive computing, the average energy efficiency is improved by 1.93× due to the great throughput improvement. Further, with the ODAP, the energy efficiency can be improved by 1.11× on average because of the parallel processing of ConvNet and FCNet/RNN. There is an exception in benchmark Yolo. As we mentioned before, since Thinker's compiler chooses TM-flow for Yolo, there is no further improvement. Finally, by adopting FPMB, the memory banks are partitioned to CONV buffer and FC buffer, which eliminate the data access conflicts. Moreover, the fused pattern-based data reuse reduces a lot of redundant memory accesses. As a result, the energy efficiency is further improved by 1.17× on average.

Table VI shows the overall performance on each benchmark. In DRAM access, we can see, the configurations account for only a tiny proportion so the configuration cost of Thinker processor is very low. Since FCNet/RNN takes a considerable portion in hybrid-NN, the amount of weights account for a great proportion in total DRAM accesses. The final energy efficiency achieved by all optimization techniques is 2.17× ∼ 2.93× higher than baseline, which proves the effectiveness of the three proposed techniques for hybrid-NNs.

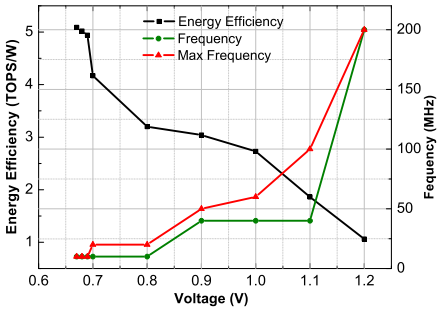


Fig. 15. Energy efficiency when voltage scales.

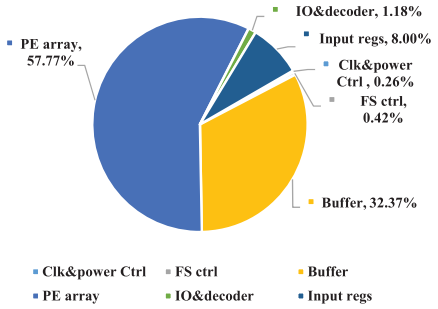


Fig. 16. Area breakdown.

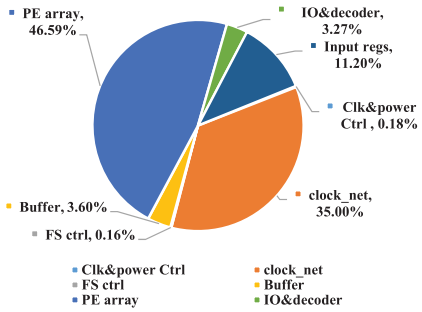


Fig. 17. Power breakdown.

5) Energy Efficiency on Voltage Scaling: In Fig. 15, we present the maximum measured energy efficiency and its corresponding working frequency. We also report the maximum working frequency with different voltage. When scaling down from 1.2 to 0.67 V, Thinker processor achieves up to 5.09 TOPS/W at 10 MHz. The power and area breakdowns are shown in Figs. 16 and 17. The power breakdown is obtained by performing post place and route simulations using actual workloads. The area breakdown is calculated based on the area report of place and route tools. We can see that the PE arrays take up 57.77% area and 46.59% power consumption, since it has a lot of registers to support different datapaths for different operations and performs all MAC operations. The buffers take up 32.37% area, but only consume 3.6% power which mainly profits from the proposed FPMB. The shared input registers also offload 11.2% power consumption from multi-bank buffers due to data reuse. The other parts show small area and power consumption.

B. Analysis of Area Efficiency and PE Array Scalability

In Thinker architecture, both general PE and super PE can perform multiple functions by reusing computing resources.

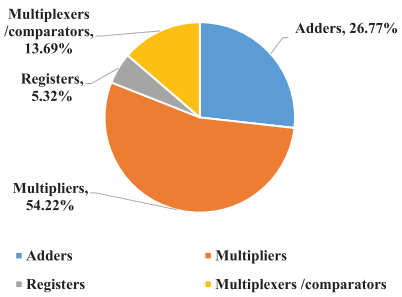


Fig. 18. General PE area breakdown.

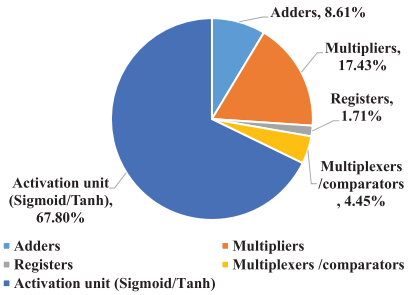


Fig. 19. Super PE area breakdown.

Figs. 18 and 19 show the area breakdown of general PE and super PE, respectively. The total gate count of general PE is 4358. In general PE, the major units are multipliers and adders which occupy 81% area. Since they can be shared by CONV operations and FC operations, the area utilizations of 16-bit mode and 8-bit mode are 91% and 90%, respectively. The major idle resources include some of the MUXes and registers. In super PE, to support sigmoid and Tanh functions in RNN, the activation unit is implemented by linear-fitting method, which contains 10 multipliers and occupies 67.8% area. Thus, the total gate count of super PE reaches 13 619. The area utilization of super PE is in the range of 2%–72%. Since the total number of super PEs is only 32, they have very little impact on the overall area utilization, compared to 480 general PEs. When NNs are processed on PE arrays, the overall area utilization is determined by the dominant CONV and FC layers. For example, when AlexNet is processed in 16-bit mode at 200 MHz, the area utilization is about 86.4%, resulting in area efficiency of 14.18 GOPS/mm^2. Compared to the state-of-the-art NN processors fabricated in 65-nm technology, Thinker processor achieves higher area efficiency than Eyeriss (6.86 GOPS/mm^2) and comparable area efficiency with DNPU (18.75 GOPS/mm^2).

Theoretically, a large PE array can provide more partitioning choices compared with a small array, which makes AP-flow reach its optimal performance due to finer granularity. However, under the limitation of on-chip memory capacity and off-chip memory bandwidth, a large array will suffer from low PE utilization rate. Therefore, a medium-size PE array would achieve the best energy efficiency. We evaluate the throughput ratios of AP-flow over TM-flow of different array sizes by simulation using AlexNet workload. As shown in Fig. 20, when the array size is smaller than 20×20, the performance

TABLE VII
COMPARISON WITH STATE-OF-THE-ART DESIGNS

Reference	Eyeriss ISSCC2016 [13]	CNN-SOC ISSCC2017 [15]	DNPU ISSCC2017 [18]	ENVISION ISSCC2017 [12]	This Work
Technology (nm)	65	28	65	28	65
Core Area (nm^2)	3.5×3.5	6.2×5.6	4.0×4.0	1.29×1.45	3.8×3.8
Voltage (V)	0.82 ∼ 1.17	0.575 ∼ 1.1	0.77 ∼ 1.1	0.65 ∼ 1.1	0.67 ∼ 1.2
Core Frequency (MHz)	100 ∼ 250	200 ∼ 1175	50 ∼ 200	25 ∼ 200	10 ∼ 200
On-chip SRAM (kB)	181.5	5625	290	144	348
Number of PEs	168	288	776	256	512
Number of DSPs	0	16	0	0	0
Peak Performance (TOPS)	0.084 @16b	0.752 @8b	1.20 @ 4b	0.076 @4b	0.410 @8b
Bit-width (bits)	16	8/16	4/8/16	4/8/16	8/16
Power Scalability (mW)	94 ∼ 450	39 ∼ —	34.6 ∼ 279	7.5 ∼ 300	4.0 ∼ 386
Energy-Efficiency Scalability (TOPS/W)	0.14 (63.2GOPS, 16b) @250MHz, 1.17V ∼ 0.25 (23.1GOPS, 16b) @100MHz, 0.82V	— ∼ 2.9 (113GOPS, 8b) @200MHz, 0.575V	1.0 (279GOPS, 16b) @200MHz, 1.1V ∼ 8.1 (280GOPS, 4b) @50MHz, 0.77V	0.26 (76GOPS, 16b) @200MHz, 1.1V ∼ 10 (75GOPS, 4b) @50MHz, 0.65V	1.06 (409.6GOPS, 8b) @200MHz, 1.2V ∼ 5.09 (20.4GOPS, 8b) @10MHz, 0.67V
Benchmark	AlexNet	AlexNet	AlexNet	AlexNet	AlexNet
Working Voltage (V)	1.0	0.575	1.1	1.0	1.2
Working Frequency (MHz)	200	200	200	200	200
Power Consumption (mW)	278 @16b	41 @8b	279 @4b	44 @4b	290 @adaptive bit-width
Throughput (GOPS)	46.2 @16b	77 @8b	1088 @4b	76 @4b	368.4 @adaptive bit-width
Energy-Efficiency (TOPS/W)	0.166 @16b	1.89 @8b	3.9 @4b	3.8 @4b	1.27 @adaptive bit-width

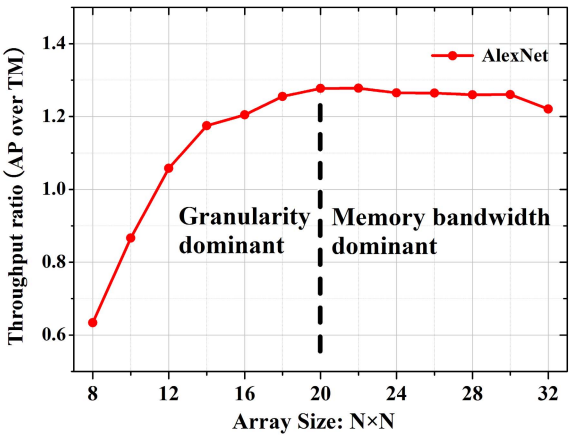


Fig. 20. Array scalability evaluated on AlexNet.

of AP-flow is granularity dominant. AP-flow's advantage rises evidently with the increasing array size and outperforms TM-flow starting from 12×12. When the array size is bigger than 20×20, the curve goes into memory bandwidth dominant region, and the advantage of AP-flow over TM-flow becomes steady.

C. Comparison With State-of-the-Art Designs

At the nominal frequency of 200 MHz with 1.2 V, Thinker processor achieves peak throughput of 409.6 GOPS and power consumption of 386 mW, corresponding to an energy efficiency of 1.06 TOPS/W. When the voltage is scaled down to 0.67 V, the throughput and power consumption are reduced to 20.4 GOPS and 4 mW at 10 MHz, corresponding to an energy efficiency of 5.09 TOPS/W.

As shown in Table VII, Eyeriss, CNN-SoC, DNPU, ENVISION, and Thinker processor are compared on the same benchmark of AlexNet, running at 200 MHz. Eyeriss achieves an energy efficiency of 0.166 TOPS/W in fixed 16-bit mode with 278-mW power consumption. CNN-SoC supports both 8-bit and 16-bit operations. With dynamic voltage and frequency scaling, it achieves at most 1.89 TOPS/W with 41-mW power consumption in 8-bit mode. Both DNPU and ENVISION support 4-, 8-, and 16-bit operations. DNPU and ENVISION achieves 3.9 and 3.8 TOPS/W with 4-bit operations, respectively. Thinker processor processes AlexNet in bit-width adaptive mode, which achieves 368.4-GOPS throughput and consumes 290-mW power, corresponding to an energy efficiency of 1.27 TOPS/W.

VI. CONCLUSION

A reconfigurable hybrid-NN processor was fabricated in 65-nm LP CMOS technology. This processor consists of two 16×16 reconfigurable heterogeneous PE arrays, which can provide 409.6-GOPS peak throughput. In order to improve energy efficiency, each PE supports bit-width adaptive computing to fully utilize computing resources, which can enhance the throughput by 1.93× on benchmarks. Meanwhile, the PE arrays can be partitioned on demand to process different NN layers in parallel, which offers a high utilization of hardware resources and achieves 11% improvement on energy efficiency. Besides, a fused pattern-based multi-bank memory system reduces a large number of redundant memory access and can flexibly provide data for PE arrays, which improves the energy efficiency by 1.17×. With all these optimization techniques, the silicon results show that this processor achieves an energy efficiency of 5.09 TOPS/W at most.

REFERENCES

[1] K. Simonyan and A. Zisserman. (2014). "Very deep convolutional networks for large-scale image recognition." [Online]. Available: http://arxiv.org/abs/1409.1556
[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2016, pp. 770–778.
[3] J. Wu, G. Wang, W. Yang, and X. Ji. (Jul. 2016). "Action recognition with joint attention on multi-level deep features." [Online]. Available: https://arxiv.org/abs/1607.02556
[4] S. Han *et al.*, "ESE: Efficient speech recognition engine with compressed LSTM on FPGA," in *Proc. ACM/SIGDA Int. Symp. FPGA*, Feb. 2017, pp. 75–84.
[5] T. N. Sainath, O. Vinyals, A. Senior, and H. Sak, "Convolutional, long short-term memory, fully connected deep neural networks," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Apr. 2015, pp. 4580–4584.

[6] J. Donahue *et al.*, "Long-term recurrent convolutional networks for visual recognition and description," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2015, pp. 2625–2634.
[7] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2015, pp. 3431–3440.
[8] J. Chen, L. Yang, Y. Zhang, M. Alber, and D. Z. Chen, "Combining fully convolutional and recurrent neural networks for 3D biomedical image segmentation," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 3036–3044.
[9] L. Kaiser *et al.* (2017). "One model to learn them all." [Online]. Available: https://arxiv.org/abs/1706.05137
[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2012, pp. 1097–1105.
[11] J. Emer, V. Sze, and Y.-H. Chen. (2017). *Tutorial on Hardware Architectures for Deep Neural Networks*. [Online]. Available: http://eyeriss.mit.edu/tutorial.html
[12] B. Moons, R. Uytterhoeven, W. Dehaene, and M. Verhelst, "14.5 Envision: A 0.26-to-10 TOPS/W subword-parallel dynamic-voltage-accuracy-frequency-scalable convolutional neural network processor in 28 nm FDSOI," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2017, pp. 246–257.
[13] Y.-H. Chen, T. Krishna, J. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2016, pp. 262–263.
[14] Y. Chen *et al.*, "DaDianNao: A machine-learning supercomputer," in *Proc. 47th Annu. IEEE/ACM Int. Symp. Microarchitecture*, Dec. 2014, pp. 609–622.
[15] G. Desoli *et al.*, "A 2.9 TOPS/W deep convolutional neural network SoC in FD-SOI 28 nm for intelligent embedded systems," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2017, pp. 238–239.
[16] S. Park, S. Choi, J. Lee, M. Kim, J. Park, and H.-J. Yoo, "A 126.1 mW real-time natural UI/UX processor with embedded deep-learning core for low-power smart glasses," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Jan. 2016, pp. 254–255.
[17] J. Sim, J. S. Park, M. Kim, D. Bae, Y. Choi, and L. S. Kim, "A 1.42 TOPS/W deep convolutional neural network recognition processor for intelligent IoE systems," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Jan. 2016, pp. 264–265.
[18] D. Shin, J. Lee, J. Lee, and H.-J. Yoo, "DNPU: An 8.1 TOPS/W reconfigurable cnn-rnn processor for general-purpose deep neural networks," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2017, pp. 240–241.
[19] N. P. Jouppi *et al.*, "In-datacenter performance analysis of a tensor processing unit," *ACM SIGARCH Comput. Archit. News (ISCA)*, vol. 45, no. 2, pp. 1–12, 2017.
[20] S. Yin *et al.*, "A 1.06-to-5.09 TOPS/W reconfigurable hybrid-neural-network processor for deep learning applications," in *Proc. IEEE Symp. VLSI Circuits (VLSI-Circuits)*, Jun. 2017, pp. C26–C27.
[21] B. Moons, B. De Brabandere, L. Van Gool, and M. Verhelst, "Energy-efficient ConvNets through approximate computing," in *Proc. IEEE Winter Conf. Appl. Comput. Vis. (WACV)*, Mar. 2016, pp. 1–8.
[22] B. Moons and M. Verhelst, "A 0.3–2.6 TOPS/W precision-scalable processor for real-time large-scale ConvNets," in *Proc. IEEE Symp. VLSI Circuits (VLSI-Circuits)*, Jun. 2016, pp. 1–2.
[23] L. Song, Y. Wang, Y. Han, X. Zhao, B. Liu, and X. Li, "C-brain: A deep learning accelerator that tames the diversity of cnns through adaptive data-level parallelization," in *Proc. 53rd ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, Jun. 2016, pp. 1–6.
[24] C. Zhang, Z. Fang, P. Zhou, P. Pan, and J. Cong, "Caffeine: Towards uniformed representation and acceleration for deep convolutional neural networks," in *Proc. Int. Conf. Comput. Aided Design (ICCAD)*, Nov. 2016, pp. 1–8.
[25] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.
[26] X. Chen. (Apr. 2015). "Microsoft coco captions: Data collection and evaluation server." [Online]. Available: http://arxiv.org/abs/1504.00325
[27] Y. Fu, E. Wu, A. Sirasao, S. Attia, K. Khan, and R. Wittig, "Deep learning with INT8 optimization on Xilinx devices," Xilinx, San Jose, CA, USA, White Paper WP486 (v1.0.1), Apr. 2017.
[28] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2016, pp. 779–788.
[29] *Caffe: Convolutional Architecture for Fast Feature Embedding*. Accessed: Jun. 2014. [Online]. Available: http://caffe.berkeleyvision.org
[30] *Image Classification and Image Captioning Performed by Thinker: A High Energy Efficient Reconfigurable Hybrid-Neural-Network Processor*. Accessed: Aug. 2017. [Online]. Available: https://youtu.be/tJ1m9B4iiK4

Shouyi Yin received the B.S., M.S., and Ph.D. degrees in electronic engineering from Tsinghua University, Beijing, China, in 2000, 2002, and 2005, respectively.

He was a Research Associate with Imperial College London, London, U.K. He is currently an Associate Professor with the Institute of Microelectronics, Tsinghua University. His current research interests include reconfigurable computing, mobile computing, and SoC design.

Peng Ouyang received the B.S. degree in electronic and information technology from Center South University, Changsha, China, in 2008, and the Ph.D. degree in electronic science and technology from Tsinghua University, Beijing, China, in 2014.

He held a post-doctoral position with the School of Information, Tsinghua University. He is currently a Researcher with the School of Electronic Information Engineering, Beihang University, Beijing, China. His current research interests include the embedded deep learning, neuron computing, and reconfigurable computing.

Shibin Tang received the B.S. degree in computer science and technology from Shandong University, Jinan, China, in 2008, and the Ph.D. degree in computer architecture from the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China, in 2014.

He is currently a Post-Doctoral Researcher with the Institute of Microelectronics, Tsinghua University, Beijing. His current research interests include computer architecture, cache coherence, on-chip memory system, deep learning, and neural network acceleration.

Fengbin Tu received the B.S. degree in electronic science and technology from the Beijing University of Posts and Telecommunications, Beijing, China, in 2013. He is currently pursuing the Ph.D. degree with the Institute of Microelectronics, Tsinghua University, Beijing.

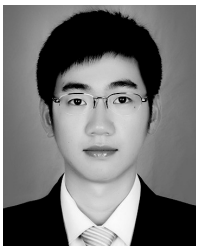
His current research interests include deep learning, computer architecture, VLSI design, approximate computing, and reconfigurable computing.

Xiudong Li received the B.S. degree in electronic information science and technology from Henan University, Kaifeng, China, in 2007, and the M.S. degree in microelectronics from the Harbin Institute of Technology, Harbin, China, in 2010.

He was an ASIC Physical Design Engineer with NVIDIA semiconductor technology, Shanghai, China. He is currently an ASIC Design Engineer with the Institute of Microelectronics, Tsinghua University, Beijing, China. His current research interests include the design and implementation of deep learning accelerator processor, mobile computing, and wireless communication.

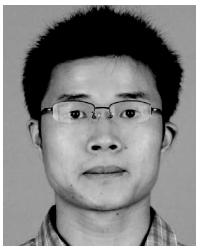
Shixuan Zheng received the B.S. degree from Tsinghua University, Beijing, China, in 2016, where he is currently pursuing the Ph.D. degree with the Institute of Microelectronics.

His current research interests include reconfigurable computing architecture and VLSI design.



Tianyi Lu received the B.S. degree from Southeast University, Nanjing, China, in 2015. He is currently pursuing the M.S. degree with the Institute of Microelectronics, Tsinghua University, Beijing, China.

His current research interests include reconfigurable computing and optimization of compiler for reconfigurable computing.



Jiangyuan Gu received the B.S. degree from the School of Microelectronics, Xidian University, Xi'an, China, in 2014. He is currently pursuing the Ph.D. degree with the Institute of Microelectronics, Tsinghua University, Beijing, China.

His current research interests include reconfigurable computing and low-powered compiling technique for reconfigurable computing.

Leibo Liu received the B.S. degree in electronic engineering from Tsinghua University, Beijing, China, in 1999, and the Ph.D. degree from the Institute of Microelectronics, Tsinghua University, in 2004.

He is currently an Associate Professor with the Institute of Microelectronics, Tsinghua University. His current research interests include reconfigurable computing, mobile computing, and VLSI DSP.

Shaojun Wei was born in Beijing, China, in 1958. He received the Ph.D. degree from the Faculte Polytechnique de Mons, Mons, Belgium, in 1991.

He was a Professor with the Institute of Microelectronics, Tsinghua University, Beijing, in 1995. His current research interests include the VLSI SoC design, EDA methodology, and communication ASIC design.

Dr. Wei is a Senior Member of the Chinese Institute of Electronics.