Addressing the Issue of Processing Element Under-Utilization in General-Purpose Systolic Deep Learning Accelerators

Bosheng Liu[†*], Xiaoming Chen[†], Ying Wang[†], Yinhe Han[†], Jiajun Li[†*], Haobo Xu[†*], and Xiaowei Li[†]
[†]State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences
[*]University of Chinese Academy of Sciences
{liubosheng, chenxiaoming, wangying2009, yinhes, lijiajun01, xuhaobo, lxw}@ict.ac.cn

ABSTRACT

As an energy-efficient hardware solution for deep neural network (DNN) inference, systolic accelerators are particularly popular in both embedded and datacenter computing scenarios. Despite their excellent performance and energy efficiency, however, systolic DNN accelerators are naturally facing a resource under-utilization problem -- not all DNN models can well match the fixed processing elements (PEs) in a systolic array implementation, because typical DNN models vary significantly from applications to applications. Consequently, state-of-the-art hardware solutions are not expected to deliver the nominal (peak) performance and energy efficiency as claimed because of resource under-utilization. To deal with this dilemma, this study proposes a novel systolic DNN accelerator with a flexible computation mapping and dataflow scheme. By providing three types of parallelism and dynamically switching among them: channel-direction mapping, planar mapping, and hybrid, our accelerator offers the adaptability to match various DNN models to the fixed hardware resources, and thus, enables flexibly exploiting PE provision and data reuse for a wide range of DNN models to achieve optimal performance and energy efficiency.

1 Introduction

Deep neural networks (DNNs) have become a hot research topic because of their powerful computing accuracy for a wide range of cognitive tasks, such as image classification, text analysis, context understanding, and so forth [1-3]. The great success of state-of-the-art DNNs largely comes from the huge amount of training data and large-scale models. However, as the widespread requirement of efficiently processing massive datasets (e.g. TB scale), speeding up computing performance and minimizing power consumption for DNN applications are highly desirable by a broad range of cognitive computing systems, from battery-powered embedded and Internet of Things devices to datacenters.

According to the hardware structure, most of application-specific integrated circuit (ASIC) based DNN accelerators can be classified into three categories: 2D [4, 5], reconfigurable [6, 7], and systolic [8-10]. Particularly, the systolic architecture, with a high-throughput array of processing elements (PEs) as the primary computing component to offer excellent data reusability, has been widely recognized as the ideal architecture for specialized DNN accelerators. By exploiting data reuse inside a systolic array, a variety of systolic accelerators have been proposed for saving memory bandwidth and enabling huge PE scale under limited hardware footprints [8, 9]. For example, Google's TPU [9] provides a superior systolic architecture with tens of thousands of PEs, which is 256x larger than the 2D accelerator, Diannao [4].

ASPDAC '19, January 21–24, 2019, Tokyo, Japan
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-6007-4/19/01…$15.00
https://doi.org/10.1145/3287624.3287638

Table 1. PE utilization in systolic accelerators.

	Hardware type	DNNs	PE utiliz.
Wei et al [10]	FPGA	a)Alexnet	40.32%
		b)VGG	89.11%
TPU [9]	ASIC	a)CNN in AlphaGo	46.2%
		b)LSTM in [9]	8.2%

Despite their feats of memory bandwidth and scalability in DNN accelerations, we have found in experiments that most state-of-the-art systolic architectures (e.g., [8, 9]) cannot always well match the computation kernels of DNNs to a fixed-size PE array, incurring vast amounts of PEs to an idle state. Table 1 demonstrates the PE utilization rates of state-of-the-art systolic architectures. It can be observed that (1) the PE under-utilization issue widely exists in systolic accelerators, and (2) the PE utilization differs significantly for different DNN models. Consequently, the inability to differently handle the net configuration results in severe PE under-utilization. For example, the idle PE rate of TPU reaches even up to 91.8% for Long Short-Term Memory (LSTM) inferences. This serious "PE utilization wall" will significantly bring down the energy-efficiency of current systolic DNN accelerators.

To better understand the reason of PE under-utilization, Fig. 1(a) illustrates a convolution computation case on a systolic PE array from [8, 9]. Loading input activations from the same channel of input feature maps (*ia*) to the PE array provides the capability to reuse the inputs in PEs. This derives from that the output activations in each output feature map (*out*) are generated by convolving the same kernel weights (*w*) across *ia*. However, when the edge size of *ia* ($E \times H$) cannot well match the fixed-size PE array, it inevitably incurs some PEs to an idle state, identified as the PE under-utilization issue. The PE under-utilization issue, deriving from mapping input activations from the same channel, is severe because (1) most edges of *ia* in DNNs are in a small size (for example, 87.5% of the edge size in Alexnet [11] is not greater than 27×27), and (2)

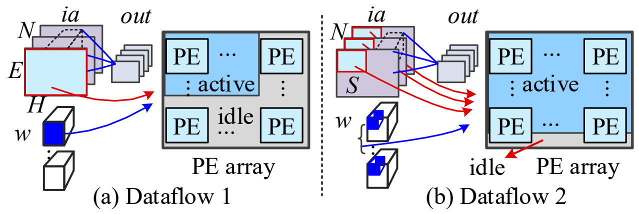


Fig 1. Motivation example. (a) Reusing activations from the same input feature map (*ia*). (b) Reusing activations from different input feature maps.

the edge size of feature maps varies significantly throughout DNN models and also from applications to applications. Prior works TPU [9] and Eyeriss [6] severely suffer from PE under-utilization, deriving from that they exploit data reuse by adding the input activations from the same channel. Though Shidiannao [8] can achieve high PE utilization with a small scale of PEs (generally below 64 PEs), its computation throughput is limited by the small PE scale, e.g., the peak performance only reaches 194 GOP/s.

The popular approach of exploiting data reuse in the same feature map induces PE under-utilization for a fixed-size PE array. For example, some input feature maps are too small to be mapped to the PE array in full utilization. However, in this case, these input feature maps often have sufficient input channels to be parallelized in the PEs, identified as **deep input layers**, such as the Conv-3 layer of AlexNet with 256 input channels but 13 activations per channel. Obviously, processing activations and weights from different channels in parallel will increase the PE utilization when inferring these deep input layers. Based on this idea, we first introduce a mapping scheme named **channel-direction mapping** to alleviate the PE utilization wall problem for deep input layers with small edge sizes. In this pattern, as shown in Fig. 1(b), the convolution operation can be performed by simultaneously loading multiple channels of inputs while exploiting data reuse among each input feature map. By providing sufficient input activations from multiple channels, the channel-direction mapping scheme outperforms prior works Shidiannao and Eyeriss in PE utilization, which load activations from the same channel. Moreover, the proposed scheme is unique to 2D architectures, such as Diannao [4], which only share channels of input activations to weights without reusing data in the PE array.

However, some feature maps are large and have few channels, called **wide input layers**, such as the Conv-1 layer of AlexNet, which has only 3 channels but a 224 × 224 size. Due to the limited channels, the proposed channel-direction mapping scheme can hardly improve the PE unitization for such layers. For this extreme case, parallelizing the computation of activations in the same input feature map will be more efficient, identified as **planar mapping** for convenience (Fig. 1(a)), which has already been deployed in Shidiannao and Eyeriss. Different from prior planar schemes, the proposed design partitions the input maps into slices according to a stride size S, and provides multiple slices of inputs from the same map to boost the PE utilization. There is no overlapping among slices, and the size of each slice is $S \times S$. In this case, the proposed design can exploit data reuse to boost the PE utilization without the impacts of different stride sizes.

Nevertheless, the above two cases are extreme cases. Many DNN layers are in a case that is between the two extremities of deep and wide input layers. Only sticking to one type of parallelization, either channel-direction mapping or planar mapping, cannot well map DNNs to the fixed PE array. Consequently, a **hybrid-mapping** is necessary, which combines the main ideas of both channel-direction mapping and planar mapping and exploits the parallelism in the two mapping dimensions. Such an extra option expands the decision space of data mapping in fixed-size PE arrays, and potentially increases the chance of achieving higher PE utilization for systolic accelerators.

By combining the above mentioned three mapping schemes together, we propose a flexible dataflow to address the PE utilization wall in systolic accelerators. The proposed design provides the adaptability of dynamically switching in the three mapping schemes to fit the shape of PE arrays, for improving the PE utilization and exploiting data reuse simultaneously. Evaluations based on six representative DNNs show that our design outperforms state-of-the-art accelerators with 1.32x~7.5x higher PE utilization and 5.7x~10.9x better energy efficiency. Specifically, we make the contributions as follows:

- We propose a flexible computation kernel mapping and dataflow for systolic accelerators to combat the found "PE utilization wall" in a wide range of DNN applications.
- We exploit three types of dataflow parallelisms, channel-direction mapping, planar mapping, and hybrid, targeting at the PE utilization and data reuse, allowing an adaptive tiling and mapping scheme for a given DNN to maximize the utilization of PEs on a systolic array.
- We implement the physical layout of the proposed design that supports the adaptive data-level parallelization.

2 Related Work

Non-systolic accelerators. There are two typical non-systolic accelerator architectures: 2D and reconfigurable. **2D architectures**: Diannao [4] uses a general-purpose 2D architecture to accelerate large-scale DNNs with high throughput. Moons et al. [5] introduces a precision-scalable 2D architecture to save the power consumption for convolutional neural network (CNN) inference. However, neither of them enables data reuse on the PE array for the PE utilization improvement. **Reconfigurable architectures**: Eyeriss [6] customizes a reconfigurable architecture that can exploit data reuse by routing input activations from the same map and shifting weights within the PE array. Tetris [7] leverages the same mapping dataflow as Eyeriss and mainly optimizes the off-chip 3D memory access for efficiency. However, reusing input activations from the same feature map will incur severe PE under-utilization, due to the small and significantly varied edge size of input feature maps. Our work can handle input activations from channels of feature maps in parallel and exploit data reuse among each input map, so that it can provide sufficient inputs for efficient PE utilization.

Systolic accelerators. The recently proposed systolic architectures have demonstrated their superior and efficient computation capability for DNN applications [8, 9]. Shidiannao [8] exploits data reuse on a systolic array with input activations from the same map to eliminate off-chip memory accesses. TPU [9] builds a superior computing array based on regular systolic interconnects for massive DNN applications in datacenters. Though the data reusability and design scalability are improved in these work, dataflow designs that achieve efficient PE utilization are not discussed. Our work provides a flexible dataflow to fully utilize channels of input activations and exploit data reuse in each channel map, so that it can provide sufficient inputs for better PE utilization and efficient accelerating computations.

3 Flexible Dataflows for Systolic Architectures

We first introduce the channel-direction mapping scheme for deep input layers and outline how we achieve data reuse and boost the PE utilization. Then we present the ideas of planar and hybrid mappings. At last, a dynamic selection of the above mappings is provided for efficient PE utilization. Because convolutional (Conv) and fully-connected (FC) layers dominate the most computations of DNNs, we focus more on the two typical layers.

A Channel-direction Mapping

A.1 Channel-direction parallelism for Conv layers

Fig. 2 depicts the channel-direction parallelism for accelerating Conv layers with deep channels on a fixed $m \times n$ (m rows and n columns) PE array. To fit the $m \times n$ PE array size, the N input channels and M output channels are partitioned into groups of size Tm and Tn ($Tm = m$, $Tn = n$), respectively (i.e., each output channel group has Tm channels and each input channel group has Tn channels), as shown in lines 2 and 4. Along the column direction of the outputs, we partition all the C output columns into groups of size Tu, as shown in line 3. The partitioning scheme indicates that we exploit input reuse within each block of outputs. Because Tu indicates the block size, it does not impact on the performance of computation, but affects the register capacity for storing outputs in the PE array. The key operations of the channel-direction parallelism include three steps. The first step is to prepare the input data. At most $Tm \times Tn$ weights and Tn input activations are provided to the $m \times n$ PE array, as shown in lines 6 and 8 respectively. All Tn input activations are shared by Tm rows of PEs. Second, we perform multiply-accumulate (MAC) operations to calculate partial sums of convolutions, as shown in line 9. At last, we add together the partial sums to generate the final results, as shown in line 12.

The proposed channel-direction mapping scheme enables to reuse weights and activations in two aspects. First, the loaded weights at line 6 can be reused Tu times by updating the input activations, as shown by the red arrow. Second, the recently loaded input activations at line 8 can also be reused through a local register file, as shown by the blue arrow. This derives from the reusable input activations in Conv layers for the adjacent Tu outputs. In this case, the recently loaded activations are temporally accommodated in the local register file. By shifting them in the register, the required activations for subsequent convolutions can be provided by the local register file instead of reloading them from on-chip memory again. The capacity of the register file should be $Tu \times Tn$, deriving from the minimal requirement of accommodating the input

```
Inputs: R: rows of output maps, M: # of output channels,
   C: columns of output maps, N: # of input channels,
   Tm, Tn, and Tu: tiling parameters, K' and K: kernel size;
1  for ro=1:R{
2  for mo=1:Tm:M{
3    for co=1:Tu:C{
4    for ni=1:Tn:N{
5    for (i=1:K',j=1:K){
         //load Tm*Tn number of weights (w)
6        loading w((mo,Tm),(ni,Tn),i,j);
7        for(co2 = co:(co+Tu-1), cur=S*(ro-1)+i){
            //add Tn activations (ia)
8           adding ia((ni,Tn),cur, (S*(co2-1)+j));
            //Mac on Tm*Tn PEs
9           sum[Tm][Tn][co2]=MAC((mo,Tm),(ni,Tn));
10   }}} //end co
11   for (tu=1:Tu, mo2=mo:(mo+Tm-1)){
        //add together Tn channels of outputs
12    out[mo2][tu]=∑(sum[mo2][*][tu])}}}}
```

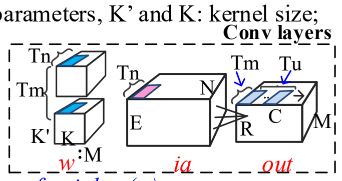


Fig 2. Channel-direction parallelization for Conv layers.

activation for reusing them for the outputs in an output block. Consequently, the channel-direction mapping scheme can fully exploit the reuse of input weights and activations in the PE array. This outperforms typical 2D architectures, such as Diannao, which share channels of input activations to weights without data reuse on the PE array.

The channel-direction parallelization scheme can achieve efficient PE utilization for Conv layers with deep channels. This is because the loaded weights and activations come from multiple channels maps. This means that, the proposed design can fully use the deep channels to provide sufficient inputs for efficient PE utilization. For most Conv layers with deep channels and small map sizes, such as the Conv-5 layer of AlexNet which has a 13 × 13 feature map size, 256 input channels and 384 output channels, the proposed design can efficiently overcome the PE under-utilization in prior accelerators, such as Shidiannao and Eyeriss, which exploit data reuse within one channel of input activations.

A.2 Channel-direction parallelism for FC layers

Fig. 3 depicts the channel-direction parallelization scheme for FC layers. Considering that the weights in FC layers are private and cannot be reused, we focus more on the activation reuse. The key operations are described as follow. First, the M output channels are partitioned into tiles of size $Tm \times Tu$, as shown in line 1, so that the input activations can be reused Tu times by updating input weights, as shown in line 4 and the red arrow. Then, the latest Tn and $Tm \times Tn$ of input activations and weights, respectively, are provided in each loading cycle, as shown in lines 4 and 6. This method ensures that we can load the same amount of activations and weights in each cycle as for Conv layers. At last, it needs to add together the MAC outputs to generate the final output activations (see line 10), since each PE contributes a partial sum of the final outputs. Based on the above key operations, the channel-direction mapping scheme for FC layers can reuse the activations under the same data provision as in Conv layers.

```
Inputs: R: rows of output maps, M: # of output channels,
    C: columns of output maps, N: # of input channels,
    Tm, Tn, and Tu: tiling parameters, K' and K: kernel size;
1  for mo = 1:(Tm*Tu):M{
2  for ni = 1:Tn:N{
3  for (i = 1:K', j = 1:K){
       //load Tn activations(ia)
4    loading ia((ni,Tn), i, j);
5    for(mo1 = 1:Tu, mo2=mo+(Tm*(mo1-1))){
         //load Tm*Tn numbers of weights (w)
6        loading  w((mo2,Tm), (ni,Tn), i, j);
         //Mac on Tm*Tn PEs
7        sum[Tm][Tn][mo1]=MAC((mo3,Tm),(ni2,Tn),i,j);
8  }}}//end ni
9  for (tu=1:Tu, mo3=mo2:(mo2+Tm-1)){
       //add together Tn channels of outputs
10   out[mo3][tu]=∑(sum[mo3][*][tu])}}
```

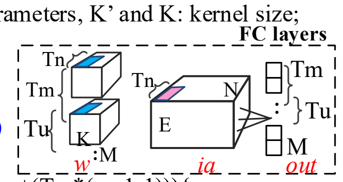


Fig 3. Channel-direction parallelization for FC layers.

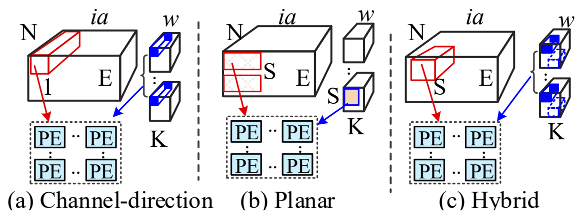


Fig 4. Planar and hybrid mappings.

The channel-direction parallelization scheme for FC layers can achieve efficient PE utilization. This is because the loaded weights and activations come from deep channels but not small-size feature maps, as shown in lines 4 and 6. Therefore, we can make full use of deep channels, such as both the input and output channels of FC-7 layer of Alexnet are 4096, to achieve efficient PE utilization in FC layers. Different from Eyeriss for FC layers that shares each input activation to all output channels of weights, the proposed scheme can fully utilize both the input and output channels of weights. Thus, the proposed scheme can achieve better PE utilization than Eyeriss for FC layers by efficiently eliminating the heavy dependence of output channels.

B. Planar and Hybrid Mappings

Though the channel-direction mapping scheme can effectively handle deep channel layers, as summarized in Fig. 4(a), it cannot well resolve wide input layers, such as the Conv-1 layer of Alexnet and Googlenet [12] with only three input channels. To resolve this issue, we introduce the planar mapping scheme to handle wide input layers. Furthermore, we propose a hybrid mapping scheme which explores a tradeoff between the two schemes and attains better PE utilization for general cases.

Fig. 4(b) depicts the planar mapping scheme for wide input layers with few input channels, especially for the case with only one input channel. It first partitions the input feature maps *ia* into slices based on the stride size *S*, so that there are no overlapped regions among slices of inputs. Data reuse and PE utilization can be exploited by adding multiple slices of activations from the same channel map to the PE array. After MAC operations, it also needs to add together the MAC outputs in the slices.

Fig. 4(c) depicts the hybrid mapping for Conv layers. It partitions *ia* into multiple equal-sized slices and stacks them up to generate more "virtual channels" for sufficient channels again. We also establish the size of each slice according to the stride size *S*. In this case, we have in total $N \times S \times S$ virtual channel maps instead of *N* (stacking up *N* channels of inputs with a $S \times S$ size). Equivalently, we convert the wide inputs into a deep input layer, to provide sufficient channels, so that we can compute them in a similar way as the deep-channel mapping scheme again for efficient PE utilization.

C. Dynamic Selection of Mapping Schemes

To maximize the PE utilization in an $m \times n$ PE array, we provide rules to select the optimal mapping scheme (channel-direction, planar, and hybrid) for DNN layers. For Conv layers with *N* input channels, *M* output channels, and stride size *S*, the selecting rules are as follows: when $M < m$, the planar scheme is optimal because the others cannot fully fill the PE array with a small *M*. In contrast, when $M > m$, there are three phases impact on the selections. First, when $N > n$ (deep channel map), the channel-direction scheme is efficient for it can provide inputs from deep channel maps. Second, when $N < n$ (wide input layer), but $N \times S \times S > n$, the hybrid scheme is effective because that the large stride can provide sufficient "virtual channel". At last, when $N < n$, and $N \times S \times S < n$, the planar scheme is efficient. Meanwhile, for FC layers, the channel-direction for FC layers (see Fig. 3) is efficient because it can reuse the input activations.

The input data layouts of the three schemes are different, and thus, once a layer is finished, we need to prepare the input data for the next layer according to the required layout of the mapping scheme. In order to achieve this and enable dynamically switching among the three types of parallelisms for different layers, we always create the correct data layout before next layer starts. The output activations in the current layer's output are dynamically reorganized after they are generated by the PE array and then stored back to the on-chip memory. Because activation writes are not on the critical path, the reorganization will not induce any performance overhead. Since our approaches are only for inference, weights are pre-trained and fixed, they are reorganized offline, which will not induce any performance overhead. Therefore, we can always correctly accommodate both activations and weights as inputs for the next layer.

4 Architecture of the Proposed Accelerator

Fig. 5 outlines the overall architecture of the proposed systolic accelerator for DNNs. It mainly consists of five components: a neural process element (NPE), an on-chip buffer including a weight buffer (WB) and two input/output activation buffers (ABin and ABout), a local register file (LRF), a direct memory access (DMA), and a central controller (CC). Specifically, NPE mainly performs MAC computations. ABin and ABout alternately store the input and output activations, while WB accommodates weights. LRF temporally stores input activations before sending them from

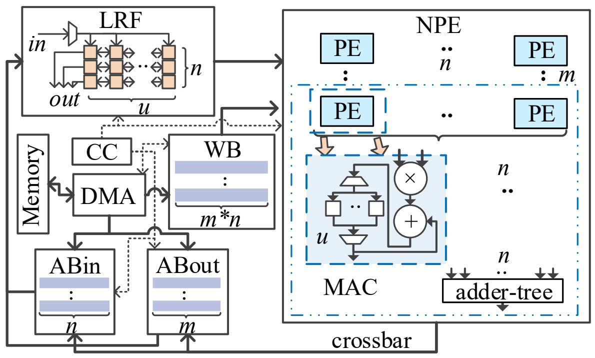


Fig 5. Architecture of the proposed systolic accelerator.

ABin/ABout to NPE and shifts them for exploiting data reuse. DMA transfers the activations and weights between on-chip buffers and the off-chip memory. CC, functioning as a controller, coordinates the computation and data movement.

NPE. NPE which consists of $m \times n$ PEs and m adder-trees with n inputs mainly performs MAC computations. Each PE is composed of a multiplier, an adder, and an output register vector with u number of registers ($u = Tu$). NPE receives $m \times n$ weights, n activations, and an index signal as inputs for multiplication and accumulation. The index signal, which comes from LRF, indicates one accumulation location of the u number of register vector. The final m output activations are generated in pipeline by the adder-trees with the data from the output register vector as inputs. After that, the m output activations will be stored back to ABout/ABin.

LRF. LRF provides the capability to exploit the reuse of recently loaded input activations. An $n \times u$ register array, shown in LRF of Fig. 5, temporally accommodates the input activations, where u is identical to the register number of the output register vector in each PE. LRF receives n numbers of activations from ABin/ABout. Then, it provides the required n activations to NPE by shifting the activations in the register array properly. The index signal can be generated by LRF for indicating the location of accumulation on PEs, which is stored in a local register. This is because the register array can use its column number as an index corresponding to the related location of input activations.

Systolic organization. NPE bonds LRF to function as a systolic organization for exploiting data reuse in DNN inference. The reused inputs on the PE array come from two aspects. First, the weights in the input register of PEs can be reused at most u times by updating input activations. Second, the input activations are reused with the help of LRF.

Parallelization transition. To enable the adaptability of the three mapping schemes, a routing unit, which is implemented by a reconfigurable multi-stage crossbar ($m \times n$ for each bit), is needed for dynamically reorganizing the output activations before storing them back to ABout/ABin. Specifically, (a) for the channel-direction mapping scheme, the generated activations are stored directly, (b) for the planar mapping scheme, activations from the same output map are grouped and reorganized to contiguous slices; and (c) for the hybrid mapping scheme, the activations of different output maps are contiguously aligned and grouped into slices before being stored. In this way, output activations can be reorganized properly at the last layer and then act as inputs for current layer according to the used mapping scheme.

5 Evaluation

Benchmarks. We use six representative neural networks, Alexnet [11], Googlenet [12], Resnet [1], Squeezenet [13], NN1024 [14], and DBN [15], to evaluate the proposed approach. Specifically, Alexnet, Googlenet, Resnet, and Squeezenet are CNNs with ImageNet as input, while NN1024 and DBN consist of only FC layers and their input is MNIST.

RTL and Layout. The proposed systolic accelerator is implemented using Verilog. And, we synthesize it with Synopsys Design Compiler. Then, we place and route it with Synopsys IC Compiler using the TSMC 65nm technology library. The testbenches for RTL simulation are created using Synopsys VCS with the actual workloads and six DNNs as inputs. The on-chip static random-access memory (SRAM) based buffers, including ABin/ABout and WB, are generated by Memory Compiler, while the off-chip memory is estimated by the Cacti [16] tool. The used total capacity of the on-chip buffers of the proposed accelerator is 160 KB (ABin: 16 KB, ABout: 16 KB, and WB: 128 KB).

Baselines and Measurements. Two famous accelerators, Shidiannao [8] and Eyeriss [6], are used as the baselines. Both of them enable the reuse of activations and weights on the PE array. Specifically, Shidiannao, a popular systolic architecture, shares each input weight to all PEs, and the input activations are reused by shifting them within the PE array. On the other hand, Eyeriss, a state-of-the-art reconfigurable accelerator for CNNs, allows data reuse in the PE array by routing input activations and shifting weights across PEs. To make the PE utilization comparison fair and adapted to various PE array sizes, we resize the PE array of Shidiannao and Eyeriss to three scales: 16x16, 32x32, and 64x64. Also, the energy efficiency, in respect to the latency and power, are evaluated with their original PE array sizes. The PE size of the proposed design is 16x16, the same as that used in [4].

A. Experiment Results

PE Utilization. To evaluate the performance of the proposed design, Fig. 6 shows the PE utilization results based on three scales of PE sizes, 16x16, 32x32, and 64x64. We observe that, on average, the proposed design (Prop) achieves 1.34x, 2.5x, and 7.5x higher PE utilization over Shidiannao on the three scales of PEs respectively. Meanwhile, we outperform Eyeriss with respectively 1.32x, 2.2x, and 7.5x higher PE utilization under the three PE sizes. The efficient PE utilization comes from two sources. First, for deep input layers, the proposed design can make full use of deep channels to boost the PE utilization. Second, for wide input layers, we can provide sufficient data provision from each input feature map and multiple channel maps.

Considering that Eyeriss and TPU have outlined their PE utilization, we give a comparison under their initial PE sizes. When comparing to Eyeriss, 168 PEs, four batch sizes, and Conv layers of Alexnet are used, which are identical to that of Eyeriss. The proposed design achieves 1.05x higher PE utilization, deriving from that we can efficiently utilize channels of inputs to overcome the unaligned edge size. Though TPU provides the PE utilization,

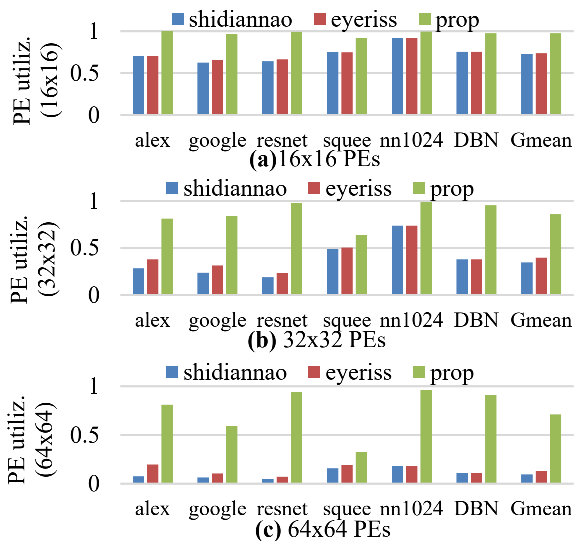


Fig 6. PE utilization comparison under three scales of the PE array size, 16x16, 32x32, and 64x64.

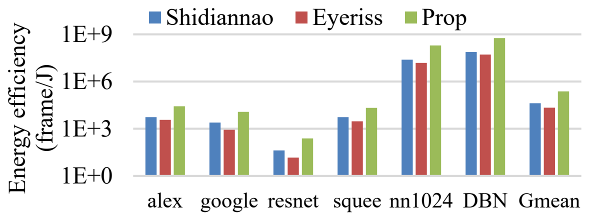
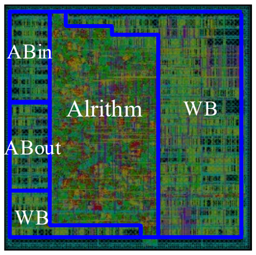


Fig 7. Energy efficiency against to Shidiannao and Eyeriss.



Component	Power (mW)	Area (mm^2)
WB	73.3(12.9%)	5.37(45.1%)
ABin	9.7(1.7%)	0.67(5.6%)
ABout	9.7(1.7%)	0.67(5.6%)
NPE	445.6(78.4%)	1.64(13.8%)
LRF	6.7(1.2%)	2e-4(0.2‰)
CC	23.4(4.1%)	0.17(1.4%)
Fill cell	-	3.38(28.4%)
Total	**568.4**	**11.9**

Fig 8. The layout achieved after placed and routed.

it is hardly to give a straightforward comparison, due to the unclear dataflow and existed pipeline hazards in TPU.

Energy efficiency. The energy consumption of the hardware accelerators is profiled under their original PE sizes. Specifically, Shidiannao, Eyeriss, and the proposed design compose of 64, 168, and 256 PEs, respectively. Fig. 7 shows the energy efficiency comparison results based on the six representative DNNs. It can be seen that, the proposed design outperforms Shidiannao and Eyeriss with average 5.7x and 10.9x better energy efficiency, respectively. The major efficiency source comes from the higher PE utilization and the powerful computation capability. Nevertheless, the larger PE size of the proposed design also means higher power consumption, reaching 1.78x and 2.0x higher than Shidiannao and Eyeriss respectively. And the used frequency of the proposed design is 0.7x and 3.5x of Shidiannao and Eyeriss respectively. Even under the same frequency and PE size as Eyeriss, we also achieve 1.2x better energy efficiency, driving from the efficient PE utilization for DNNs.

Layout and Overhead. Fig. 8 shows the layout and the power and area overheads of the proposed design. The proposed accelerator, contains 256 PEs (*Tm* = 16, *Tn* = 16, and *Tu* = 4) and 16-bit fixed-point arithmetic, achieving 537.6 GOP/s peak performance, 568.4 mW power, and 11.9 mm^2 area at 700 MHz operation frequency. It can be seen that NPE and WB dominate in the total power and area (78.4% and 45.1%) respectively. The total power and area overheads of the routing unit for reorganizing output activations are 0.19mW and 0.03mm^2, respectively (the overheads have been considered as a part of NPE), which are respectively 0.03% and 0.25% of the total power and area.

6 Conclusion

In this paper, we introduce a novel systolic DNN accelerator with three types of data-level parallelization: channel-direction mapping, planar mapping, and hybrid. The proposed design leverages adaptive data mapping to boost PE utilization. We in addition implement a flexible systolic hardware accelerator for highly efficient execution of DNN applications. Our extensive evaluations show that the proposed accelerator achieves significant benefits in PE utilization and energy efficiency compared with prior works.

ACKNOWLEDGMENTS

The work was supported in part by National Natural Science Foundation of China (NSFC) (61804155, 61522406, 61834006, 61532017, 61521092), Beijing Municipal Science &Technology Commission (Z171100000117019, Z181100008918006), and Strategic Priority Research Program of the Chinese Academy of Sciences (XDPB12).

REFERENCES

[1] K. He, et al., "Deep residual learning for image recognition," in *CVPR*, 2016, pp. 770-778.
[2] M. L. Seltzer, et al., "An investigation of deep neural networks for noise robust speech recognition," in *ICASSP*, 2013, pp. 7398-7402.
[3] X. Zhang, et al., "Character-level convolutional networks for text classification," in *NIPS*, 2015, pp. 649-657.
[4] T. Chen, et al., "Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," in *ASPLOS*, 2014, pp. 269-284.
[5] B. Moons and M. Verhelst, "A 0.3–2.6 TOPS/W precision-scalable processor for real-time large-scale ConvNets," in VLSI-Circuits, 2016, pp. 1-2.
[6] Y.-H. Chen, et al., "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," in *ISCA*, 2016, pp. 367-379.
[7] M. Gao, et al., "Tetris: Scalable and efficient neural network acceleration with 3d memory," *ASPLOS,* pp. 751-764, 2017.
[8] Z. Du, et al., "ShiDianNao: Shifting vision processing closer to the sensor," in *ISCA*, 2015, pp. 92-104.
[9] N. P. Jouppi, et al., "In-datacenter performance analysis of a tensor processing unit," in *ISCA*, 2017, pp. 1-12.
[10] X. Wei, et al., "Automated systolic array architecture synthesis for high throughput CNN inference on FPGAs," in *DAC*, 2017, pp. 1-6.
[11] A. Krizhevsky, et al., "Imagenet classification with deep convolutional neural networks," in *NIPS*, 2012, pp. 1097-1105.
[12] C. Szegedy, et al., "Going deeper with convolutions," in *CVPR*, 2015, pp. 1-9.
[13] F. N. Iandola, et al., "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 1MB model size," *arXiv preprint arXiv:160207360,* 2016.
[14] N. Srivastava, et al., "Dropout: a simple way to prevent neural networks from overfitting," *J. Machine Learning R.,* vol. 15, pp. 1929-1958, 2014.
[15] G. E. Hinton, et al., "A fast learning algorithm for deep belief nets," *Neural computation,* vol. 18, pp. 1527-1554, 2006.
[16] N. Muralimanohar, et al., "Optimizing NUCA organizations and wiring alternatives for large caches with CACTI 6.0," in *MICRO*, 2007, pp. 3-14.