Accelerating Generative Neural Networks on Unmodified Deep Learning Processors—A Software Approach

Dawen Xu, Cheng Liu®, Member, IEEE, Ying Wang®, Member, IEEE, Kaijie Tu®, Bingsheng He, Member, IEEE, and Lei Zhang, Member, IEEE

Abstract—Generative neural network is a new category of neural networks and it has been widely utilized in many applications such as content generation, unsupervised learning, segmentation, and pose estimation. It typically involves massive computing-intensive deconvolution operations that cannot be fitted to conventional neural network processors directly. However, prior works mainly investigated specialized hardware architectures through intensive hardware modifications to the existing deep learning processors to accelerate deconvolution together with the convolution. In contrast, this article proposes a novel deconvolution implementation with a software approach and enables fast and efficient deconvolution execution on the existing deep learning processors. Our proposed method reorganizes the computation of deconvolution and allows the deep learning processors to treat it as the standard convolution by splitting the original deconvolution filters into multiple small filters. Compared to prior acceleration schemes, the implemented acceleration scheme achieves $2.4 \times -4.3 \times$ performance speedup and reduces the energy consumption by $27.7\% -54.5\%$ on a set of realistic benchmarks. In addition, we have also applied the deconvolution computing approach to the off-the-shelf commodity deep learning processors. The performance of deconvolution also exhibits significant performance speedup over prior deconvolution implementations.

Index Terms—Generative neural network, deconvolution accelerator, split deconvolution

1 INTRODUCTION

Deep neural networks are making continuous breakthroughs in massive research territories over the years. In contrast to the conventional convolutional neural networks heavily utilized for object classification and detection, generative neural networks [1] have been proved to be superior in a broad domain of applications including content-generation, unsupervised learning, segmentation and pose estimation. Typically, the generative neural networks involve both convolutional layers and deconvolutional layers. Both layers are compute-intensive, but deconvolutional layers become the performance bottleneck in many cases. As presented in Table 1, deconvolution takes up a large number of multiply-and-add operations in the benchmark and is used as an indispensable component in common architectures in generative networks and other popular models such as semantic segmentation and instance detection [2]. Therefore, it is demanded to accelerate deconvolution particularly on end-devices for real-time and low-power applications such as real-time deepfake [3] and style transfer [4].

Hardware specialization is a popular approach to accelerating the computation of neural network-based applications. To accelerate generative neural networks with customized hardware, researchers have tried several approaches from distinct angles. An intuitive solution is to reuse the convolution processor and build a unified fully convolutional processor for both convolution and deconvolution operations. In such architectures, input data of deconvolution can be reorganized by dynamically padding zero activations to the original feature maps and the deconvolution can be treated as the conventional convolution as presented in Fig. 1. Fig. 1a is an example of the classic deconvolutional operation with the stride of 2. Fig. 1b is a converted equivalent convolutional operation with the stride of 1. With this conversion, the deconvolution can be mapped to the convolution processor without any hardware modification. However, the zero activations induce considerable redundant computing and degrade the performance as illustrated in [5]. Although many CNN accelerators [5], [6], [7], [8], [9], [10] are able to skip the zero activations during the computing through additional zero detection logic, they typically can only skip a portion of the zero activations especially the ones that are located on the boundary of the feature maps. The zero-padding deconvolution approach as shown in Fig. 1b has many
zero activations inserted between the non-zero activations and they are usually difficult to be removed due to the aligned data flows on the parallel computing units in deep learning accelerators.

To improve the computing efficiency of deconvolution, the authors in [11] opted to build independent processor engines for convolution and deconvolution operation respectively. This approach raises a large portion of hardware resources and chip area increase. Different from the above two approaches, the authors in [5] and [12] proposed to revisit the convolutional processor and change the micro-architecture to support both convolution and deconvolution efficiently in a unified processor. In addition, these methods also need a dedicated data flow scheduler to make use of the computing engine. For unified architectures, the advantage is better performance and hardware utility, while the disadvantage is the additional redesign and engineering cost. However, for off-the-shelf CNN processors without specialized deconvolution support such as DianNao [7] and TPU [13], the inefficiency and resource under-utilization induced by the zero-padding approach is an inevitable cost to implement the deconvolutional layers.

Inspired by the prior works, we seek to support fast and efficient deconvolution layer implementation on general CNN processors like Eyeriss [6], DianNao [7] and TPU [13], some of which are already commercialized and widely used in different areas. For these classic CNN processors, many zero-value activations must be padded to the feature map in order to map the deconvolution layers on to them. Instead of zero-padding that induces numerous redundant computing operations, we tailor a novel implementation of deconvolution layer from the software angle, and pre-partition the deconvolutional filters into multiple small convolutional filters, so that the deconvolution operations are converted and can be efficiently implemented on any CNN processor without redesigning or replacing them. In our evaluation of classic CNN processors, the performance and the energy efficiency of our deconvolution implementation remains competitive compared to prior work of specialized GAN processors.

In summary, our contributions can be summarized as follows:

- We proposed a novel filter partitioning and reorganization approach to convert a general deconvolution operation to multiple standard convolution operations strictly without incurring much computing redundancy such that deconvolution can be implemented efficiently as convolution.
- We investigated the way to reorganize the split deconvolution results efficiently on legacy neural network processors without hardware modification.

The rest of this paper is organized as follows. Section 2 presents the related work of deconvolution acceleration design. Section 3 describes the architecture of typical CNN processors. In Section 4, we elaborate on the conversion process of generic split deconvolution in detail. At length, Section 5 presents the evaluation results and Section 6 concludes the paper.

2 Related Work

With the advancements of deep learning, various neural networks have been proposed to address different tasks such as objection detection and image classification. Among them, generative neural networks are demonstrated to be particularly efficient for content generation tasks like image style transfer [2], [4], [14], segmentation tasks such as [15], and pose estimation tasks such as [16]. These novel neural networks attract a lot of attention. Ledig et al. [17] proposed SRGAN and adopted a perceptual similarity loss to generate detailed images from low-resolution images. By using generative adversarial networks (GANs), high-resolution images of small objects can be generated and utilized to improve target detection accuracy [18]. Generative neural networks can also be applied for sequence data generation as presented in SeqGAN [19] and ORGAN [20]. Additionally, more variants of generative neural networks have been developed and employed in semi-supervised learning and the medical field [21], [22].

However, generative neural networks that consist of both compute-intensive convolution and deconvolution operators cannot be fitted to the conventional CNN processors directly [6], [7], [8], [23]. As deconvolution is also computing-intensive and hinders the acceleration of generative neural networks on CNN processors, thereby, it is highly demanded to explore hardware acceleration of deconvolution operations. Zhang et al. in [11] proposed to optimize deconvolution with reverse looping and stride hole skipping. Despite the excellent performance on a set of representative benchmarking networks with comprehensive experiments, the experiments show that the proposed approach achieves competitive performance over the state-of-the-art deconvolution processors on both general CNN processors and the most advanced commodity deep learning processor chips such as Google TPU and Intel Neural Compute Stick 2, which are released recently.

Table 1 shows the number of multiply-add operations in the inference phase on different benchmarks. The table shows the total number of operands and the number of deconvolution operands, as well as the ratio of deconvolution operations. The benchmarks include DCGAN [33], ArtGAN [14], SNGAN [34], GP-GAN [35], MDE [4], and FST [3].

Table 1: The Number of Multiply-Add Operations in the Inference Phase

| Benchmarks   | Total operands (M) | Deconvolution operands (M) | Ratio   |
|--------------|--------------------|----------------------------|---------|
| DCGAN [33]  | 111.4              | 109.8                      | 98.6%   |
| ArtGAN [14] | 1268.8             | 822.1                      | 64.8%   |
| SNGAN [34]  | 100.9              | 100.7                      | 99.8%   |
| GP-GAN [35] | 240.4              | 103.8                      | 43.2%   |
| MDE [4]     | 2638.2             | 849.4                      | 32.2%   |
| FST [3]     | 94730.5            | 604.0                      | 0.6%    |

Fig. 1. Computational process of (a) deconvolution and (b) deconvolution with inserted zero-values.
performance, combining independent convolution and deconvolution components in a processor induces considerable chip area and power consumption. Amir Y et al. in [12] proposed to convert deconvolution to convolution by adding zeros to the activations and then developed a unified MIMD-SIMD processor for both operations. In addition, it implemented a set of distributed on-chip buffers to avoid the redundant computing brought by the inserted zero activations. Based on [12], the authors further developed an end-to-end template-based solution, which can generate the optimized synthesizable unified processor from a high-level specification of GANs in [24]. Instead of adding zeros to the input feature map, Xu et al. in [5] proposed a unified FCN processor on top of a bi-direction systolic array. The FCN processor performs the computing on original input features. The weight and data of adjacent PEs are shared and passed periodically by taking advantage of the small column buffers added to the 2D PE array. Similar to [5], Wang et al. in [25] designed a uniform architecture to support both 2D and 3D deconvolutional neural networks on FPGAs. Multiple FIFOs are added to adjacent PEs to deliver the overlapped temporary results. Yan et al. in [26] proposed a cold buffer to store the overlapped computing results for more efficient data reuse and a novel mapping approach to improve the utilization of the computing array for both convolution and deconvolution. Intel NCS2 [27] also has specialized hardware to support native deconvolution, but there are no open technical details. The authors in [28] proposed a filter deformation method to eliminate all the inserted zero activations when deploying the deconvolution on ReRAM-based neural accelerators. It supports both inference and training with high resource utilization, but it is specialized for the 3D ReRAM architectures instead of the conventional CMOS-based neural accelerators. In addition, the deformation method leads to imbalanced data flow and requires dedicated mapper unit and scheduler unit to manage the data flow in ReRAM-based accelerators. In summary, it can be found that hardware redesigning is typically required to have an existing CNN processor to support deconvolution computing in generative neural networks. Due to the long hardware design cycle, many commodity neural network processor including Google Edge TPU [13] and Ropal Neural Compute Stick Lightspeeur SPR2801 [29] still do not support the raw generative neural networks yet.

Different from the above works, researchers seek to reuse the conventional CNN processors for generative neural networks without hardware redesigning. Shi et al. [30] presented a simple example of transformation from deconvolution to convolution by padding zeros to the input feature maps. However, the fixed zero-padding to the right and bottom of the input features only works for the first partition of the split deconvolution and it can cause errors when this zero-padding is utilized for the deconvolution conversion. The correct padding must be adapted to the deconvolution partition as well as the output feature cropping strategies to ensure equivalent output to the raw deconvolution. Besides, this work is posted as a blog with limited experiments and has not gone through the peer review. Chang et al. [31] utilized filter deformation and proposed an approximate conversion approach targeting at super-resolution image reconstruction problems. While super-resolution image reconstruction can typically tolerate computing errors, the approximate conversion approach works fine but it cannot be applied to general generative neural networks that are not necessarily fault-tolerant. In addition, the approach proposed in [31] needs to rearrange the deconvolutional results on CPU instead of CNN processors and it can cause massive data communication between CPU and the processors. To address the above problems, we aim to develop a software approach that can deploy generative neural networks directly on the existing CNN processors without precision penalty nor hardware modification.

3 Typical CNN Processors

This section briefly explains the architecture of the two mainstream architectures of general CNN processors assumed in this paper, including dot-production array processor and regular 2D array processor. Most of the prior CNN processors can be categorized into these two kinds of typical architectures [5], [6], [7], [8], [9], [10], [13].

3.1 Dot-Production Array Processor

Fig. 2 shows the dot-production based CNN processor. It consists of $D_{out}$ neural processing units. Each neural process unit includes $D_{in}$ multipliers as well as an adder tree and performs a dot production. The same $D_{in}$ input activations are fed concurrently to each processing unit per cycle while the weights are different. Each unit accepts $D_{in}$ weights per cycle and $D_{in} \times D_{out}$ parameters need to be sent to the array per cycle. In each PE, $D_{in}$ partial results obtained from multipliers are consumed by the adder tree, and a dot production can be completed each cycle because of the pipelined processing architecture. Once a filter window is processed, the output result is sent to an activation function unit and the result is transferred to the output buffer. Each output activation produced by the processing unit belongs to a different output channel. When weight or data cannot be accommodated by the on-chip buffers, the neural networks will be tiled to fit the architecture. Diannao [7], Dadiannao [8], C-brain [9] and Cnvlutin [10] are typical designs that adopt the dot-production array architecture.

3.2 Regular 2D Array Processor

Another typical CNN processor architecture with regular 2D PE array is illustrated in Fig. 3. Compared to the former
structure, it mainly differs from the data flow. The data flow used in this work is output stationary (OS) according to the definition in Eyeriss [6]. Basically, each PE in the array performs all the operations required to yield an output activation. The weights are fed from the first column of the array and flow across the PEs from left to right to guarantee that all PEs operate in full scale. The input activations are broadcast to all the PEs in a column, but we have at most one PE column to receive the input activations alleviating the pressure on on-chip buffers bandwidth. Each row of the PE array produces output activations of one output feature map on the y-axis. Each column PE produces the output activations belonging to different output feature maps but the same pixel positions. Under the circumstances, both input activations and weights consume a limited amount of on-chip memory bandwidth. This architecture enables the proposed system to achieve high reusability and eventually benefits more on boosting throughput in a limited bandwidth provision. This feature is handy in reducing the bandwidth demand caused by weight/activation load and store, especially when the processors are sharing the on-chip storage space and bandwidth with other application processors in nowadays heterogeneous SoCs adopted in mobile and embedded systems. Eyeriss [6], TPU [13], FCN-Engine [5] are typical designs that adopt the 2D array architecture.

Algorithm 1. Convolution & Deconvolution

Require: \(o_{h}, o_{w}, o_{c}, I_{C}, K_{H}, K_{W}\)
Ensure: output \(o_{h}, o_{w}, o_{c}\)
1: function CONV
2: \(\text{for} \ (i_{c} = 0; i_{c} < I_{C}; i_{c}++) \ \text{do} \)
3: \(\text{for} \ (k_{h} = 0; k_{h} < K_{H}; k_{h}++) \ \text{do} \)
4: \(\text{for} \ (k_{w} = 0; k_{w} < K_{W}; k_{w}++) \ \text{do} \)
5: \(\text{output}(o_{h}, o_{w}, o_{c}) += \text{input}(o_{h} \times s + k_{h}, o_{w} \times s + k_{w}, i_{c}) \times w(k_{h}, k_{w}, i_{c}, o_{c}) \)
6: function DECONV
7: \(\text{left} = \text{max}(0, \text{ceil}((o_{w} - K_{W} / s)\))
8: \(\text{right} = \text{min}(I_{W} - 1, \text{left} + \text{ceil}(K_{W} / s))\)
9: \(\text{top} = \text{max}(0, \text{ceil}((o_{h} - K_{H} / s))\)
10: \(\text{bottom} = \text{min}(I_{H} - 1, \text{top} + \text{ceil}(K_{H} / s))\)
11: \(\text{for} \ (i_{c} = 0; i_{c} < I_{C}; i_{c}++) \ \text{do} \)
12: \(\text{for} \ (i_{h} = \text{top}; i_{h} < \text{bottom}; i_{h}++) \ \text{do} \)
13: \(\text{for} \ (i_{w} = \text{left}; i_{w} < \text{right}; i_{w}++) \ \text{do} \)
14: \(\text{output}(o_{h}, o_{w}, o_{c}) += \text{input}(i_{h}, i_{w}, i_{c}) \times w(o_{h} - i_{h} \times s, o_{w} - i_{w} \times s, i_{c}, o_{c}) \)

Fig. 3. Regular 2D array CNN processor [5], [6], [13]

4 THE PROPOSED SPLIT DECONVOLUTION

In Section 4.1, we analyze the correlation between the convolution and deconvolution and brief the idea of converting a deconvolution operation to generic convolutions. Then we present the detailed conversion steps from generic deconvolution operations to standard convolution operations in Section 4.2.

4.1 Correlation Between Convolution and Deconvolution

The computing patterns of convolution operation and deconvolution operation are shown in Figs. 4a and 4b respectively. For the convolution, each output feature data is the accumulation of the dot production of the filter window \(K_{H} \times K_{W}\) and the input feature window in different channels. By sliding the window across the input feature with all the filters, the entire output feature can be calculated. Note that \(I_{H}, I_{W},\) and \(I_{C}\) stand for the input feature height, width and channel size while \(O_{H}, O_{W},\) and \(O_{C}\) stand for the output feature height, width and channel size accordingly. For the deconvolution, each input feature data is multiplied to a filter first. Then the result windows generated from different input feature data are placed based on the stride (s) of the deconvolution denoted with s to form the output feature. When the result windows are overlapped, the corresponding data will be accumulated as the output feature data. Otherwise, the data in the result windows will be used as the output feature data directly. From the perspective of the computing patterns, convolution and deconvolution are completely different.

In order to reuse the conventional CNN processors for deconvolution operations, we seek to convert the computing pattern of deconvolution to that of convolution. To begin, we analyze the calculation of a single output feature...
With this observation, we proposed a split deconvolution approach which divides the deconvolution filters into multiple smaller convolution filters with the same stride \( s \) as shown in Fig. 4c. The conversion is formulated in Algorithm 2. It can be observed that the weights used for each convolution are scattered over the deconvolution filters and the deconvolution filters can be split into multiple convolution filters. In this case, the output data of each convolution that are calculated with the scattered deconvolution weights correspond to scattered output of the original deconvolution accordingly. Therefore, the output of the different convolution need to be reorganized to obtain the expected deconvolution output.

**Algorithm 2: Split Deconvolution**

Require: \( o_h, o_w, o_c, I_C, K_H, K_W \)

Ensure: output \( (o_h, o_w, o_c) \)

1: function SPLITDECONV\( (o_h, o_w, o_c, I_C, K_H, K_W) \)
2: // weights of the \( N \) identical split
3: // convolution: \( w( K_{TH}, K_{TW}, I_C ) \)
4: for \( n = 0; n < N; n++ \) do
5: for \( i_c = 0; i_c < I_C; i_c++ \) do
6: \( k_{ih} \leftarrow K_{TH} - 1 \)
7: for \( k_h = \left( \lceil n/s \rceil \right) ; k_h < K_H; k_h + s \) do
8: \( k_{iw} \leftarrow K_{TW} - 1 \)
9: for \( k_w = n \% s ; k_w < K_W; k_w + s \) do
10: \( w(n, k_{ih}, k_{iw}, i_c) \leftarrow w(k_h, k_w, i_c) \)
11: \( k_{iw} \leftarrow k_{iw} - 1 \)
12: \( k_{ih} \leftarrow k_{ih} - 1 \)
13: \( \text{Conv}(\delta_{k_h}, \delta_{k_w}, \delta_{i_c}, I_C, K_{TH}, K_{TW}) \)
14: Reorganize the obtained \( n \)th output activation.

The second step is to split the expanded deconvolution filters (the length and width are \( P_K \) ) into multiple small filters \( (K_T \times K_T) \) with sampling and rotation. Fig. 6 illustrates the coordinate distribution of filters before and after the conversion with a small but representative example. To compute an output deconvolution activation with standard convolution operations, filters need to be sampled with stride \( s \) and reorganized into new filters. In addition, each sampled filter needs to be rotated 180 degrees to ensure
correct computing. Equation (3) presents the generic conversion. Each deconvolution will be split into \( s' \) convolution operations. The stride of the split convolution operations is constant 1. Without loss of generality, suppose \( W_n \) is the \( n \)th convolutional filter. It can be obtained with Equations (4), (5), (6), (7), and (8) where \( W \) is the deconvolution filter, \((y, x)\) is the original filter coordinate and \((y_n, x_n)\) is the new coordinate.

\[
N = s^2
\]

\[
n = s \times \text{mod}(y, s) + \text{mod}(x, s)
\]

\[
W_n(y_n, x_n) = W(y, x)
\]

\[
\begin{cases}
    x_n = K_T - \text{ceil}(x / s) \\
y_n = K_T - \text{ceil}(y / s)
\end{cases}
\]

where

\[
\begin{cases}
    0 \leq x < K + P_K \\
    0 \leq y < K + P_K \\
    0 \leq x_n < K_T \\
    0 \leq y_n < K_T
\end{cases}
\]

\[
n \in \{0, 1, 2, \ldots, N - 1\}
\]

Step 1 and Step 2 basically split the deconvolution filters to multiple small convolution filters. This needs to be done only once and can be reused. Therefore, they can be done off-line with a software approach. Unlike the first two steps, Step 3 and 4 are performed on the CNN processors for each input feature map. In step 3, the input feature maps also need to be padded with zeros to obtain equivalent deconvolution output. Otherwise, the output activations on the edge will be ignored. \( P_f \) columns/rows of zeros will be added where \( P_f \) is obtained from:

\[
P_f = K_T - 1.
\]

Finally, as shown in the fourth step in Fig. 5, the \( N \) split convolution outputs need to be reorganized to form the deconvolution output. The detailed reorganization approach is illustrated in Fig. 7 and formulated in Equations (10), (11), (12), and (13). In this case, \( N \) is set to be 4 and the stride \( s \) is 2. Contrary to the filter splitting process, we select an output feature data from each split convolution output to construct an \( s \times s \) window in the expected deconvolution output. Note that \((x^n_i, y^n_i)\) represents the \( i \)th coordinates of split convolution output where \( n \) is given by Equation (4) and \((x_f, y_f)\) refers to the coordinates of expected deconvolution output. The correspondence relation between these coordinates is presented by Equations (10) and (11).

\[
x_f = x^n_i \times s + \text{mod}(n, s)
\]

\[
y_f = y^n_i \times s + \text{floor}(n / s)
\]

where

\[
\begin{cases}
    0 \leq x_i < I + 2P_f - K_T + 1 \\
    0 \leq y_i < I + 2P_f - K_T + 1
\end{cases}
\]

The output reorganization is conducted while writing the convolution output to external memory via a DMA module. As shown in 7, the sequential output of each convolution is stored in scattered locations of the memory, which is essentially stride write. Since stride memory access is usually supported in DMA module, output reorganization can be done by general neural accelerators without hardware modification. When the output data is organized in row-major or column-major, the stride write drops by 1/2 compared to sequential write because only 1/2 data are valid for each DRAM write. To ensure the output reorganization efficiency, we have the convolution output data stored in channel-major and a line of data in channel dimension is written per stride. In this case, a single write is usually long enough to make use of the DRAM bandwidth, the overall output organization is close to sequential memory bandwidth and will not incur performance penalty.

With the above four steps, we can convert generic deconvolution operations to split convolution operations and apply deconvolution on an unmodified CNN processor. Despite the hardware compatibility, the proposed split deconvolution approach may extend the filters and input feature maps, which will induce additional computing overhead. On the other hand, the padded zeros can be potentially skipped by the conventional CNN processor optimizations. The detailed evaluation of realistic benchmarks will be discussed in the experiments.

5 EXPERIMENTS

This section consists of three parts. First, we listed the setting of the selected benchmarks and the experimental environment. Then we evaluate the performance and energy consumption of split deconvolution on the generic propose processors. At last, the approach is compared with two off-the-shelf processors i.e., Google Edge TPU and Intel NCS2. The proposed SD algorithm and its deployment on the neural network processors are open-sourced and can be found in https://github.com/wangying-ict/FCN.

5.1 Experimental Setup

To perform a comprehensive evaluation of the proposed split deconvolution computing approach, we conduct experiments
on both simulation-based neural network processors and commodity neural network processors provided by the chip vendors and then compare proposed methods with prior deconvolution computing approaches.

Simulation-Based Neural Network Processors. For the simulation-based evaluation, we developed cycle-accurate neural network simulators for both the dot-production based neural network processor architecture and the regular 2D array architecture. Both the 8-bit dot-production PE array and the 2D PE array are implemented and synthesized with Synopsys Design Compiler (DC) under the TSMC 40nm library. The dot-production based architecture includes 16 processing units, and each unit performs dot production on 16 input activations and weights. The 2D PE array is set to be 32 by 7. The I/O buffer size is set to be 256 KB, weight buffer is 416 KB. Both processors run at 800 MHz.

As zero padding is required to convert deconvolution to convolution, zero skipping optimizations that can be used to improve the computing efficiency of the accelerators is beneficial to the performance of the split deconvolution. There have been many different sparse optimization approaches proposed to skip zeros in activations (Asparse), weights (Wsparse) or both activations and weights (AWsparse) [6], [10], [32]. These relatively general optimizations implemented in accelerators can be used to perfectly remove the zeros brought by the proposed split deconvolution because this approach only adds entire zero rows or columns to the input features or weights which is easy to detect and skip during the processing. While the processor with dot-production PE array utilized in this work does not support zero weight-skipping and we only apply the Asparse method on it.

Commodity Neural Network Processors. For the commodity neural network processors, we choose two representative ones. One of them is Edge TPU [13] from Google and it does not support native deconvolution operations. To implement deconvolution on it, we convert the deconvolution to standard convolution using zero-padding [6]. The other processor chip is the latest NCS2 [27] from Intel. It supports native deconvolution operation and the deconvolution is applied directly to the optimized architecture of NCS. The performance on the commodity processors is measured using the system clock.

Benchmarks. To evaluate the different deconvolution approaches, we selected a set of advanced neural networks as our benchmarks including ArtGAN [14] on Cifar 10 (ArtGAN), DCGAN [33] on Large-scale CelebFaces Attributes Dataset (DCGAN), Spectral Normalization for GAN [34] on Cifar 10 (SNGAN), GP-GAN on Transient Attributes Database [35] (GP-GAN) for generating new datasets. Unsupervised Monocular Depth Estimation of FCN on KITTI and Cityscapes [2] (MDE) aims of image segmentation and Fast-Style-Transfer [4] on CoCo2014 which is used to apply the style of one image to another image (FST).

5.2 Experimental Results on General CNN Processors

This section illustrates how the proposed split deconvolution improves the performance and efficiency of generative neural networks on the simulated general CNN processors including both dot-production array and 2D array architectures.

### Table 2

| Benchmarks | Original Deconvolution (M) | Naive Zero-padding Deconvolution (M) | Split Deconvolution (M) |
|------------|---------------------------|-------------------------------------|------------------------|
| DCGAN [33] | 109.8                     | 439.1                               | 158.1                  |
| ArtGAN [14] | 822.1                     | 2030.0                              | 822.1                  |
| SNGAN [34] | 100.7                     | 402.7                               | 100.7                  |
| GP-GAN [35] | 103.8                     | 415.2                               | 103.8                  |
| MDE [4]    | 849.3                     | 3397.4                              | 1510.0                 |
| FST [3]    | 604.0                     | 2415.9                              | 1073.7                 |

5.2.1 Operation Number and Parameters Comparison

Multiply-add (MAC) operation takes up the majority of the computing in neural networks, so the number of MACs exhibits the computing intensity of the neural networks directly and it is independent with the underlying computing architectures. Thereby, we use this metric to compare the different deconvolution computing approaches. Table 2 shows the number of MACs in original neural networks, neural networks using native zero padding (NZP) and neural networks using the proposed split deconvolution (SD). It can be observed that NZP incurs a large number of redundant operations compared with the original deconvolution. Compared to NZP, SD brings in much less computing. It does not incur any additional computing overhead in SNGAN, ArtGAN and GP-GAN and induces only a portion of additional computing on the rest of the neural networks. In theory, SD will not increase the amount of the computation, but there are occasions that zeros need to be added for more efficient computing on processors. When the original filter length or width is not divisible by the stride s in the according neural networks, we need to pad zeros on the top and left side of the filters to ensure identical filter splitting. In addition, the split deconvolution may produce only the center area of the original deconvolution output feature maps, and we must add zero padding to the input feature maps to obtain equivalent deconvolution output feature maps. Thereby, the proposed split deconvolution may add zeros to both the weights and the input activations, and induce more computing depending on the neural network parameters.

5.2.2 Preprocessing Overhead of Split Deconvolution

The proposed split deconvolution requires offline preprocessing to obtain the reorganized convolution filters. To evaluate the overhead of the preprocessing, we measured the runtime using the deconvolution benchmark on a desktop computer equipped with Intel i7-7700 3.60 GHz and 16 GB DRAM running Ubuntu 16.04. Then we compared the offline preprocessing time with the model compilation time as listed in Table 3. The comparison reveals that the preprocessing is orders of magnitude faster than the compilation. Thereby, it incurs little negative influence on the overall model compilation nor execution.

5.2.3 Performance Comparison

In this section, we mainly compare the different deconvolution approaches on typical neural network processors.
Although NZP and SD may induce redundant computing, many of the redundant computing can be potentially squeezed using the sparse aware optimization techniques which allow the processors to skip the zero multiplications. Three different sparse-aware optimization methods including Asparse, Wsparse and AWsparse are applied. In addition, we also compare with FCN-engine [5] that had the 2D PE array CNN processor redesigned. Fig. 8 depicts the normalized performance of three acceleration schemes on the dot-production PE array. NZP incurs 75 percent computing redundancy on average on the benchmark neural networks when converting the deconvolution to convolution. Unlike the NZP, split deconvolution has only marginal zero pad-dings on the boundary in some corner cases. Therefore, it has much less computing redundancy, which is projected in the 2.5/C2 performance boost of SD over NZP. When the specified input activation lines can be skipped to generate standard deconvolution output, the performance can further be improved. Notably, SD-Asparse on DCGAN improves by 1.4X. The primary reason lies in the fact that the DCGAN has fewer network layers and smaller input feature maps. As a result, the computing redundancy caused by the padding affects the overall performance more significantly. On the 2D PE array CNN processor as shown in Fig. 9, SD-Asparse and SD-Wsparse in the experiments show the influence of the filter expansion and the input expansion respectively. Although SD-Wsparse induces some redundant computation due to padding to the input feature maps, most of the convolution processors support zero-skipping and can squeeze the computing redundancy automatically. Compared to SD-Wsparse, SD-Asparse that enables the zero-skipping reduces 22 percent redundant computation on average. Similarly, SD-Asparse has zero-padding added to the weights, and the redundant computing can also be eliminated on a sparse convolution processor architecture. For workloads like DCGAN, FST and MDE, the filters need to be expanded. In these cases, SD-WAsparse reduces 75 - 80 percent computing redundancy with zero-skipping. When the split deconvolution is deployed on optimized CNN processors, the performance of SD-WAsparse is on par with that of FCN in all the benchmark neural networks. The deconvolution approach presented in FCN-engine [6] adopts a bi-directional data flow. It has implemented the original deconvolution, which is the input activations multiplied with each filter and then accumulates the overlapped production. By taking advantage of the column buffers, it can transmit the partial results for accumulation efficiently. However, the output feature maps on edge are redundant and need to be cropped, which inevitably induces computing overhead, especially for smaller deconvolution layers. Therefore, SD-WAsparse outperforms FCN-engine on some of the neural networks like DCGAN, as shown in Fig. 9.

### 5.2.4 Energy Consumption Comparison

Figs. 10 and 11 present the relative energy consumption distribution of the different deconvolution approaches on the dot-production PE array and regular 2D PE array respectively. Compared to NZP, the average energy consumption of SD-Asparse and SD-WAsparse reduce by 36.15 and 43.63 percent respectively on the two CNN architectures. Unlike the performance comparison, the energy consumption comparison is less significant. In general, the deconvolution energy consumption roughly consists of three parts i.e., PE, on-chip buffer and DRAM. According to the estimation using CACTI [36], the energy is mostly consumed by the DRAM access and the on-chip buffer access. While the amount of DRAM access of the different deconvolution approaches is about the same, their consumption has little difference across these approaches. Despite the dramatic difference in PE activity and energy consumption, PE energy

| Benchmarks | Total Compilation Time (s) | Split Time (s) | Ratio |
|------------|-----------------------------|----------------|-------|
| DCGAN [33] | 6.897                       | 0.042          | 0.611 |
| ArtGAN [14] | 23.505                      | 0.216          | 0.92% |
| SNGAN [34] | 11                          | 0.049          | 0.67% |
| GP-GAN [35] | 7.663                       | 0.053          | 0.69% |
| MDE [4]    | 21.573                      | 0.190          | 0.88% |
| FST [3]    | 3.942                       | 0.003          | 0.08% |
consumption is too small to affect the overall deconvolution energy consumption. As a result, the energy consumption difference is primarily determined by the amount of on-chip buffer accesses, which explains all the energy consumption difference. For example, SD-Asparse induces relatively more weight reading and thus higher energy consumption. Similarly, FCN requires additional on-chip buffers to support the unified convolution and deconvolution, so the overall energy consumption is higher than that of SD-WAsparse in all the benchmark networks, though their performance is quite close to each other.

5.2.5 SD-based DCGAN Demo

As we need to manipulate the output write instruction of the neural network processor to reorganize the split deconvolution outputs for the equivalent deconvolution output, we apply the proposed SD approach on an edge AI system of which we can touch the low-level output write instructions to demonstrate the use of the proposed SD algorithm. Note that the edge AI is produced by [37]. It consists of both RISC-V cores and a neural network processor fabricated with TSMC 40nm technology. Each split convolution is executed sequentially on the neural network processor and the conventional sequential output write instruction is replaced with a stride write instruction which is widely supported in DMA cores. On this AI system, we implemented a face generation demo using DCGAN as shown in Fig. 12. The end-to-end performance comparison with NZP is consistent with that obtained in Fig. 9, which demonstrates the computing efficiency of SD on neural network processors.

5.2.6 Deconvolution Conversion Quality Evaluation

In this section, we mainly evaluate the quality of the deconvolution results calculated using different deconvolution conversion approaches. We compare the results to that generated from the raw deconvolution with SSIM metric [38] which is widely utilized to measure the similarity between images. SSIM ranges from 0 to 1 and higher SSIM indicates a higher similarity between the images. The comparison is shown in Table 4. It can be observed that SD produces identical results for both DCGAN and FST, while the methods proposed in [30] and [31] produce different results and the SSIM of the same deconvolution conversion approach varies on different generative neural network models. Particularly, the approach in [30] results in considerable computing errors in DCGAN while minor computing errors in FST. This is mainly caused by the fact that the input images in FST are larger and the influence of the wrong padding on the boundaries is less significant. Moreover, the proportion of deconvolution in FST is smaller than that in DCGAN, which also explains the higher SSIM metric in FST.

To further illustrate the effect of the errors on the generated images, we also display the images generated using different deconvolution conversion approaches in Figs. 13 and 14. It can be seen that the quality of the images calculated with the different deconvolution conversion approaches is roughly consistent with the SSIM metric. Basically, FST using the deconvolution conversion approach proposed in [30] seems to be acceptable visually. However, the generated images in the rest cases differ dramatically and can not be tolerated or utilized.

5.3 Experiments on Commodity NN Processors

This section shows the use of SD for generative neural networks on the most advanced commodity CNN processor chips including Google Edge TPU without specialized deconvolution support and Intel NCS2 with specialized deconvolution operation support. As we cannot touch the internal output write instructions in these commodity neural network processors, we can not reorganize the split convolution results for the equivalent deconvolution outputs.
directly though the stride output write is probably supported. To demonstrate the use of SD on these neural network processors, we move the results generated in each split convolution to host and have the host to reorganize the results for the following neural network operations. Since the data movement is not required given internal data movement support in the neural network processors, we only take the split deconvolution computing time and the data reorganization time as the overall deconvolution execution time in the experiments.

5.3.1 Edge TPU

Edge TPU is a tensor processor with the systolic array architecture and is usually used as a co-processor of a host computer. It does not support native deconvolution operation, so we apply the NZP approach to implement the deconvolution on it as the baseline. Meanwhile, we also perform split deconvolution (SD) to deploy deconvolution on TPU. The NZP approach requires zero-padding to the input feature maps, and the SD approach needs additional output feature reorganization. While this computing cannot be performed on TPU directly, we have them done on the host processor.

The normalized acceleration performance of the two deconvolution approaches on Edge TPU is illustrated in Fig. 15. The proposed SD achieves 1.5× performance speedup over NZP on average. Particularly, FST yields the highest speedup (1.7×) over NZP on Edge TPU. However, the performance improvement is much lower than that on CPU, which is not consistent with the number of operations as listed in Table 2. To explore the underlying reasons, we further evaluate the computing efficiency of convolution with different feature map sizes and filter sizes on Edge TPU. The evaluation result is revealed in Tables 5 and 6. The filter size is set to be 3 × 3, which is a frequent setup for split deconvolution and measure the Giga multiply-add operations per second (GMACPS) given different input feature maps. As shown in Table 5, when the size of input feature maps ranges from 8 × 8 to 128 × 128, the normalized computational efficiency of Edge TPU i.e., GMACPS increases significantly.

It can be found that the proposed SD achieves 3.0× performance speedup over NZP on average, which is roughly consistent with the magnitude of operation reduction presented in Table 2. Particularly, the performance speedup goes up to 3.6× on GP-GAN. Similar to Fig. 9, the average performance improvement of DCGAN, FST and MDE are relatively lower than that of NGAN, ArtGAN and GP-GAN due to the additional parameters padded to the filters and the input features during splitting. This confirms the analysis that SD does reduce the amount of computing compared to that in NZP but the converted convolution with smaller kernel sizes and lower computing efficiency affects the performance speedup. If the neural network processors improve its computing efficiency for smaller convolution kernel sizes, the performance speedup of SD over NZP will be higher accordingly.

| Filter Size | # of input channels | # of output channels | Normalized GMACPS |
|-------------|---------------------|----------------------|-------------------|
| 2x2         | 256                 | 128                  | 1x                |
| 3x3         | 256                 | 128                  | 2.2x              |
| 4x4         | 256                 | 128                  | 3.8x              |
| 5x5         | 256                 | 128                  | 5.7x              |

Table 5: Normalized GMACPS for Different input Feature Map Size on Edge TPU

| Filter Size | # of input channels | # of output channels | Normalized GMACPS |
|-------------|---------------------|----------------------|-------------------|
| 2x2         | 256                 | 128                  | 1x                |
| 3x3         | 256                 | 128                  | 2.2x              |
| 4x4         | 256                 | 128                  | 3.8x              |
| 5x5         | 256                 | 128                  | 5.7x              |

Table 6: Normalized GMACPS for Different Filter Size on Edge TPU
5.3.2 Intel Neural Compute Stick 2 (NCS2)

NCS2 is a neural network processor produced by Intel, and it includes specialized hardware to support native deconvolution operation. We evaluated the deconvolutional layers of generative neural networks on it with the deconvolution operations implemented using the NZP approach, the SD approach as well as the native deconvolution. The experiment is presented in Fig. 17. When compared to NZP, the proposed SD performs $1.7\times$ performance speedup over the NZP approach. Similarly to Edge TPU, its performance speedup is lower than that analyzed with MACs. Therefore, we also evaluate the influence of different feature map size and filter size and the result is shown in Tables 7 and 8 with the same configurations as Edge TPU. And we notice that the lower computing efficiency of smaller convolution feature maps on NCS2 is the major reason for the lower performance speedup.

While NCS2 also includes specialized hardware for native deconvolution operation, we further evaluated the deconvolutional layers of generative neural networks on it with the optimized deconvolution. Even compared to the native deconvolution implementation on NCS2, the proposed SD approach still yields $1.1\times$ performance speedup on average. Despite the degraded computing efficiency of NCS2 on the split convolution kernels, the proposed SD approach still shows higher performance NCS2 without any hardware modification.

6 CONCLUSION

Prior generative neural network acceleration may either require intensive hardware modification of existing CNN processors or incur considerable redundant computation when converting deconvolution to convolution with straightforward zero padding, because the involved deconvolution operations cannot be fitted to the conventional CNN processors directly. To address this problem, we propose to convert the deconvolution to standard convolution with a software approach. The basic idea is to investigate the computing patterns of deconvolution and formulate it as convolution computing patterns. The resulting convolution filters can be obtained by splitting the original deconvolutional filters while the convolution results need to be reorganized to construct the original deconvolution results. This approach incurs little computing redundancy and thus enables fast and efficient deconvolution execution on legacy deep learning processors. With comprehensive experiments, we demonstrate that SD achieves $2.4\times - 4.3\times$ performance speedup over the naive zero-padding methods and is on par with the prior optimized implementation on modified fully convolution neural network processor. Moreover, the proposed approach is also beneficial to commodity neural processors. It yields $1.5\times$ performance speedup compared to the naive zero padding on Google Edge TPU which does not have native deconvolution support. When compared to Intel NCS2 chips with native deconvolution support, it still achieves $1.1\times$ performance speedup on average though the computing efficiency of NCS2 degrades with the split convolution kernels.

ACKNOWLEDGMENTS

This work was supported in part by National Science Foundation of China under Grant 61874124, and Grant 61902375, in part by Strategic Priority Research Program of Chinese Academy of Sciences under Grant XDC05030201, in part by Youth Innovation Promotion Association of Chinese Academy of Sciences under Grant 2018138, in part by MoE AcRF Tier 1 Grant of Singapore (T1 251RES1824) and Tier 2 Grant of Singapore (MOE2017-T2-1-122). Dawen Xu and Cheng Liu contributed equally to this work.
et al. "Generative adversarial nets," in Proc. Adv. Neural Inf. Process. Syst., 2014, pp. 2672–2680.

C. Godard, O. Mac Aodha, and G. J. Brostow, “Unsupervised monocular depth estimation with left-right consistency," in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2017, pp. 270–279.

I. Kourounova, W. Shi, J. Dambre, and L. Theis, “Fast face-swap using convolutional neural networks," in Proc. IEEE Int. Conf. Comput. Vis., 2017, pp. 3677–3685.

L. Engstrom, “Fast style transfer," 2016. [Online]. Available: https://github.com/lengstrom/fast-style-transfer/

D. Xu, K. Tu, Y. Wang, C. Liu, B. He, and H. Li, “FCN-engine: Accelerating deep convolutional layers on classic CNN processors," in Proc. Int. Conf. Comput.-Aided Des., 2018, Art. no. 22.

Y.-H. Chen, J. Emerson, and V. Sze, “Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," ACM SIGARCH Comput. Architecture News, vol. 44, no. 3 pp. 367–379, 2016.

T. Chen, et al., “DianNao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," ACM Sigplan Notices, vol. 49, no. 1, pp. 21–24, 2014.

Y. Chen et al., “DaDianNao: A machine-learning supercomputer," in Proc. 47th Annu. IEEE/ACM Int. Symp. Microarchitecture, 2014, pp. 609–622.

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/ IEEE Conf. Comput. Des., 2015, pp. 600–612.

J. Albericio, P. Judd, T. Hetherington, T. Aamodt, N. E. Jerger, and A. Moshovos, “Cnvlutin: Ineffecutal-neuron-free deep neural network computing," ACM SIGARCH Comput. Architecture News, vol. 44, no. 3, pp. 1–13, 2016.

X. Zhang, S. Das, O. Neopane, and K. Kreutz-Delgado, “A design methodology for efficient implementation of deconvolutional neural networks on an FPGA," CoRR, vol. abs/1705.02583, 2017. [Online]. Available: http://arxiv.org/abs/1705.02583

A. Yazdanbakhsh, K. Samadi, N. S. Kim, and H. Esmaeilzadeh, “GANAX: A unified mim-d2md acceleration for generative adversarial networks," in Proc. 45th Annu. Int. Symp. Microarchitecture, 2018, pp. 650–661.

J. Li, X. Liang, Y. Wei, T. Xu, J. Feng, and S. Yan, “Perceptual generative adversarial networks for small object detection," in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2017, pp. 1222–1230.

L. Yu, W. Zhang, J. Wang, and Y. Yu, “SeqGAN: Sequence generative adversarial nets with policy gradient," in Proc. 31 AAAI Con. Artif. Intell., 2017, pp. 2852–2858.

G. L. Guzmaraes, I. Sanchez-Lengeling , P. L. C. Farias, and A. Aspuru-Guzik , “Objective-reinforced generative adversarial networks (organ) for sequence generation models," CoRR, vol. abs/1705.10843, 2017. [Online]. Available: http://arxiv.org/abs/1705.10843v2

L. Chongxuan, T. Xu, J. Zhu, and B. Zhang, “Triple generative adversarial nets," in Proc. Advances Neural Inf. Process. Syst., 2017, pp. 4646–4658.

D. Yang et al., “Automatic vertebra labeling in large-scale 3D ct using deep image-to-image network with message passing and sparsity regularization," in Proc. Int. Conf. Inf. Process. Med. Imaging, 2017, pp. 633–644.

Y. Wang, J. Xu, Y. Han, H. Li, and X. Li, “Deepburning: automatic generation of FPGA-based learning accelerators for the neural network family," in Proc. 53rd Annu. Des. Autom. Conf., 2016, Art. no. 110.

A. Yazdanbakhsh, et al., “FlexiGAN: An end-to-end solution for FPGA acceleration of generative adversarial networks," in Proc. IEEE 26th Annu. Int. Symp. Field-Programmable Custom Comput. Mach., 2018, pp. 65–72.

D. Wang, J. Shen, M. Wen, and C. Zhang, “Towards a uniform architecture for the efficient implementation of 2D and 3D deconvolutional neural networks on FPGAs," in Proc. IEEE Int. Symp. Circuits Syst., 2019, pp. 1–5.

J. Yan, S. Yin, F. Tu, L. Liu, and S. Wei, “CNA: Reconfigurable and efficient architecture for generative network acceleration," IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., vol. 37, no. 11, pp. 2519–2529, 2018.

“Intel neural compute stick 2," 2019. [Online]. Available: https://software.intel.com/en-us/neural-compute-stick

F. Chen, L. Song, H. Li, and Y. Chen, “Zara: A novel zero-free dataflow accelerator for generative adversarial networks in 3d space," in Proc. 56th ACM/IEEE Des. Autom. Conf., 2019, pp. 1–6.

“Ropal neural compute stick, lightspeeur spr2801," 2019. [Online]. Available: https://www.ropal.com.cn/

W. Shi et al., “Is the deconvolution layer the same as a convolutional layer?” CoRR, vol. abs/1609.07009, 2016. [Online]. Available: http://arxiv.org/abs/1609.07009

J.-W. Chang and S.-J. Kang, “Optimizing FPGA-based convolutional neural networks accelerator for image super-resolution," in Proc. 23rd Asia South Pacific Des. Autom. Conf., 2018, pp. 343–348.

S. Zhang, Z. Du, Z. Lei, H. Lan, and Y. Chen, “Cambricon-x: An accelerator for sparse neural networks," in Proc. IEEE/ACM Int. Symp. Microarchitecture, 2016, pp. 1–12.

A. Radford, L. Metz, and S. Chintala, “Unsupervised representation learning with deep convolutional generative adversarial networks," CoRR, vol. abs/1511.06434, 2015. [Online]. Available: http://arxiv.org/abs/1511.06434

T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida, “Spectral normalization for generative adversarial networks," CoRR, vol. abs/1802.05957, 2018. [Online]. Available: http://arxiv.org/abs/1802.05957

H. Wu, S. Zheng, J. Zhang, and K. Huang, “GP-GAN: Towards realistic high-resolution image blending," CoRR, vol. abs/1703.07195, 2017. [Online]. Available: http://arxiv.org/abs/1703.07195

S. Li, K. Chen, J. H. Ahn, J. B. Brockman, and N. P. Jouppi, “CACI-P: Architecture-level modeling for SRAM-based structures with advanced leakage reduction techniques," in Proc. Int. Conf. Comput.-Aided Des., 2011, pp. 694–701.

“jeepio IoT chip j2," 2018. [Online]. Available: https://jeepio.com/

Z. Wang et al., "Image quality assessment: from error visibility to structural similarity," IEEE Trans. Image Process., vol. 13, no. 4, pp. 600–612, Apr. 2004.

Dawn Xu received the BS degree in computer science from Xidian University, Xi’an, China, in 2007, and the MS and PhD degrees from the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China, in 2009 and 2013, respectively. He is currently an associate professor with the Hefei University of Technology, Hefei, China. His current research interests include heterogeneous computing, VLSI design and testing, and reliable system.

Cheng Liu (Member, IEEE) received the BEng and MEng degrees from the Harbin Institute of Technology, Harbin, China, in 2007 and 2009 respectively, and the PhD degree from the University of Hong Kong, in 2016. Currently, he is an associate professor with the Institute of Computing Technology, Chinese Academy of Sciences. His research interests include FPGA-based reconfigurable computing and customized hardware acceleration.
Ying Wang (Member, IEEE) received the BS and MS degrees in electronic engineering from the Harbin Institute of Technology, Harbin, China, in 2007 and 2009, respectively, and the PhD degree in computer science from the Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS), Beijing, China, in 2014. He is currently an associate professor with ICT, CAS. His current research interests include computer architecture and VLSI design, specifically memory system, on-chip interconnects, resilient and energy-efficient architecture, and machine learning accelerators.

Kaijie Tu received the BS and MS degrees from the Hefei University of Technology, Hefei, Anhui, China, in 2016 and 2019. He is currently an assistant engineer of the Research Center for Ubiquitous Computing Systems, Institute of Computing Technology, Chinese Academy of Sciences, Beijing. His research falls primarily with the field of computer architecture, with an emphasis on the acceleration of deep neural networks.

Bingsheng He (Member, IEEE) received the bachelor’s degree in computer science from Shanghai Jiao Tong University, in 2003, and the PhD degree in computer science from the Hong Kong University of Science and Technology, in 2008. He is an associate professor with the School of Computing of National University of Singapore. His research interests include high performance computing, distributed and parallel systems, and database systems.

Lei Zhang (Member, IEEE) received the PhD degree in computer architecture from the Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS), Beijing, China, in 2008. He is currently an associate professor with the State Key Laboratory of Computer Architecture, ICT, CAS. His current research interests include network-on-chip, 3-D integration, fault-tolerant computing, and multicore/many-core processors.

For more information on this or any other computing topic, please visit our Digital Library at www.computer.org/csdl.