HNPU: An Adaptive DNN Training Processor Utilizing Stochastic Dynamic Fixed-Point and Active Bit-Precision Searching

Donghyeon Han, *Graduate Student Member, IEEE*, Dongseok Im, *Graduate Student Member, IEEE*, Gwangtae Park, *Graduate Student Member, IEEE*, Youngwoo Kim, *Graduate Student Member, IEEE,* Seokchan Song, *Graduate Student Member, IEEE*, Juhyoung Lee, *Graduate Student Member, IEEE*, and Hoi-Jun Yoo, *Fellow, IEEE*

***Abstract*— This article presents HNPU, which is an energy-efficient deep neural network (DNN) training processor by adopting algorithm-hardware co-design. The HNPU supports stochastic dynamic fixed-point representation and layer-wise adaptive precision searching unit for low-bit-precision training. It additionally utilizes slice-level reconfigurability and sparsity to maximize its efficiency both in DNN inference and training. Adaptive bandwidth reconfigurable accumulation network enables reconfigurable DNN allocation and maintains its high core utilization even in various bit-precision conditions. Fabricated in a 28-nm process, the HNPU accomplished at least 5.9× higher energy efficiency and 2.5× higher area efficiency in actual DNN training compared with the previous state-of-the-art on-chip learning processors.**

***Index Terms*— Adaptive bandwidth reconfigurable accumulation network (AB-RAN), deep neural network (DNN), in–out slice skipping (IOSS), layer-wise adaptive precision search, online learning, slice-level sparsity exploitation, stochastic dynamic fixed point.**

I. Introduction

TRAINING deep neural network (DNN) requires a significant amount of computation, which is possible only on cloud servers. For this reason, most of the DNN accelerators [1]–[6] for edge or mobile applications have supported only DNN inference. Nowadays, on-chip learning processors for edge or mobile applications are receiving increasing attention due to their ability to various applications.

First, it has a key role in distributed learning. Server-centric DNN training has a limited network scale-up, but it can be resolved by spreading the training workload across multiple edge devices. Moreover, federated learning [7], one of the typical distributed learning, can protect privacy by avoiding the upload of the user's private data to the server.

The second application is personalization, which can provide user-specific services. The on-chip learning processors fine-tune a pre-trained network to recognize local data of the individual user environment. It can increase the recognition accuracy and also recognize new objects or categories that are customized by the user.

Finally, on-chip learning can be an energy optimization method by continuously compensating for accuracy degradation. Mobile-oriented DNNs show less computation amount and memory access. However, they can be used in limited applications and show poor performance compared with the huge-size DNNs. On-chip learning can give mobile-oriented networks the opportunity to be utilized in practical situations. For example, a previous on-chip learning processor [8] utilized a light object recognition network, but it could have the object tracking functionality by adopting online DNN training. In this case, DNN training compensated for accuracy degradation that occurred by deformation of object shape or illumination changes during the tracking. The other example is temporal knowledge distillation (TKD) [9], which improves the detection performance by using online learning. Since it rarely uses the huge teacher network and utilizes a small student network most of the time, it gives another energy-efficiency improvement solution to the DNN system.

Even though on-chip learning has big advantages, the implementation of an on-chip learning processor on the edge and mobile platforms is extremely challenging. Fig. 1 explains two main difficulties that should be solved to realize the DNN training in mobile devices. First, the on-chip learning needs high throughput (>3 TOPS) because it requires not only feed-forward (FF) but also error-propagation (EP) and weight gradient (WG) stages. Generally, the DNN training is based on mini-batch gradient descent, and it utilizes multiple inputs as a batch and repeats the three training stages over multiple iterations. Therefore, the required amount of operations for the DNN training can be exponentially increasing according to the number of batches and iterations. Most on-chip learning processors [10]–[13] concentrated to improve the throughput by skipping zeros induced by the ReLU activation function,

Manuscript received November 22, 2020; revised February 8, 2021; accepted March 6, 2021. Date of publication March 23, 2021; date of current version August 26, 2021. This article was approved by Associate Editor Vivek De. This work was supported by the Institute for Information and Communications Technology Promotion (IITP) Grant funded by the Korea Government (MSIP) (On-Device Instant Learning Complex Intelligence Processor Architecture and ADAS Design for Self-driving Platform) under Grant 2019–0-01372. *(Corresponding author: Hoi-Jun Yoo.)*

The authors are with the School of Electrical Engineering, Korea Advanced Institute of Science and Technology (KAIST), Daejeon 34141, South Korea (e-mail: hdh4797@kaist.ac.kr; hjyoo@kaist.ac.kr).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/JSSC.2021.3066400.

Digital Object Identifier 10.1109/JSSC.2021.3066400

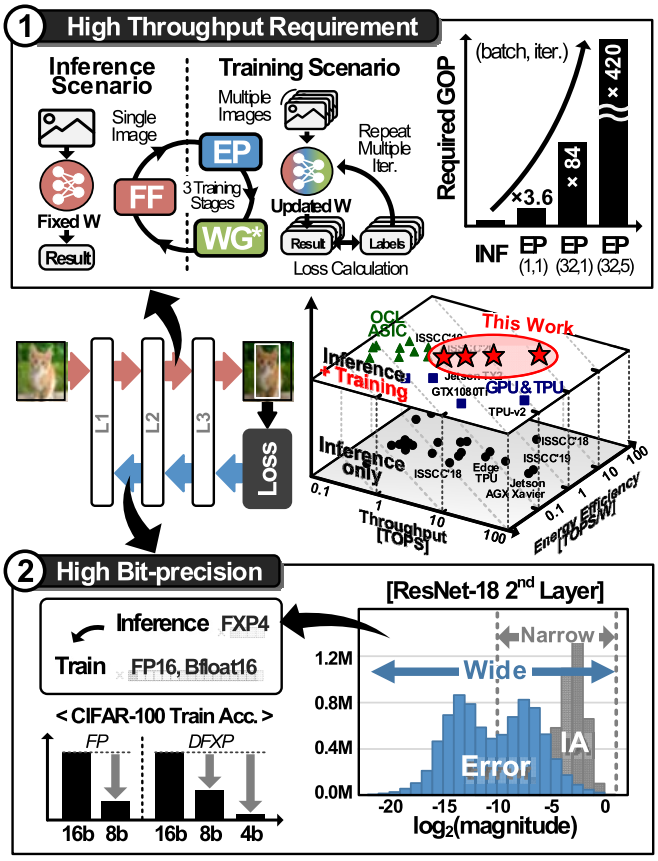


Fig. 1. Main difficulties of on-chip learning.

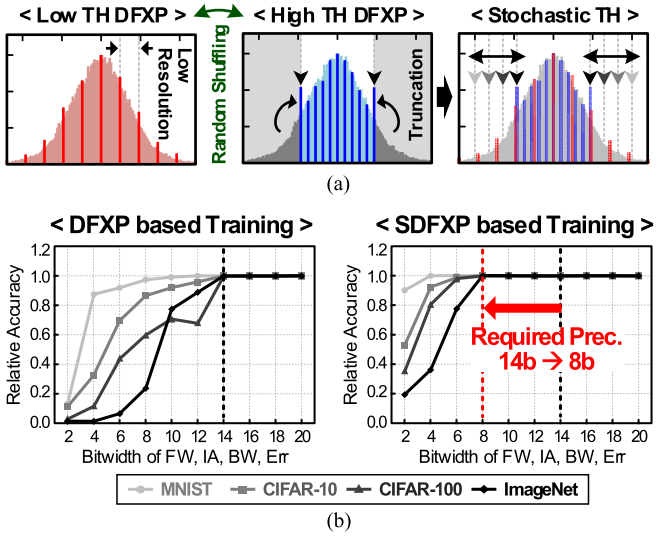


Fig. 2. Stochastic dynamic fixed-point-based DNN Training (a) Outline of ST. (b) SDFXP based bit-precision reduction result.

but the effect is minor because only 40%-to-60% sparsity is appeared by ReLU in practical usages.

Second, DNN training needs high bit-precision. Unlike the input activation (IA) that appeared in the FF stage, the range of back-propagation error is much wider, so high-bit-precision error representation is necessary for accurate training. Since the DNN training is sensitive to accuracy degradation, at least 16-bit floating-point (FP) representation has been considered for the previous on-chip learning processors. The high-bit-precision requirement of the DNN training results in low energy efficiency for the previous on-chip learning processors. The LNPU [10] suggests FP16-FP8 mixed-precision computing to release this problem, but it still needs high-bit-precision multiply-and-accumulation (MAC) units. The area and energy efficiencies of the mixed-precision FP MAC unit are much lower than the previous fixed-point (FXP)-based bit-scalable processors [4], [5].

The proposed processor, HNPU, adopts algorithm-hardware co-optimization to achieve energy-efficient DNN training by solving these difficulties. The HNPU enables low-bit-precision DNN training by introducing a new number representation, stochastic dynamic FXP (SDFXP). It can find the optimal precision by introducing layer-wise adaptive precision scaling (LAPS). Its main core adopts bit-slice serial architecture to minimize useless computations during the LAPS-based DNN training. Even though a certain layer requires high bit-precision, both sparsity-aware input-slice skipping (ISS) and precision-aware output-slice skipping (OSS) techniques compensate for throughput and energy-efficiency degradation. Finally, an adaptive bandwidth reconfigurable accumulation network (AB-RAN) gives DNN allocation reconfigurability while maintaining high core utilization even in various bit-precision conditions.

We organized the rest parts of this article as follows. In Section II, conventional low-bit-precision training methods are summarized and new low-bit-precision training algorithms, SDFXP and LAPS, are introduced in Section III. Section IV describes the overall architecture of the HNPU. Section V introduces three architecture and circuit-level features: 1) the bit-slice serial architecture with the SDFXP and LAPS units; 2) in–out slice skipping (IOSS) core architecture; and 3) AB-RAN. Section VI shows chip implementation results, and this article will be concluded in Section VII.

II. Conventional Low-Bit-Precision Training Methods

Before explaining the details of HNPU, conventional low-bit-precision training methodologies will be summarized. We will deal with FP- and FXP-based DNN training and related low-bit-precision training methodologies.

A. Floating-Point-Based Training

Conventional DNN training processors generally adopted FP-based training because of its wide numeric range. Both LNPU and GANPU [11] adopted FP16 training, but LNPU also utilized FP8 for energy-efficient DNN training. The LNPU adopted a fine-grained mixed-precision scheme to represent outliers with a sparse FP16 number format. There were also some trials to suggest new FP representations such as bfloat16 [14], Flexpoint [15], and DLfloat [16] to support energy-efficient on-chip learning. In addition, a recent algorithm [17] insisted on FP8-only DNN training, but accumulation was still done with high bit-precision such as FP16.

B. Fixed-Point Representation-Based Training

FXP-based computing has not been considered for training because of its limited numeric range and resolution. There were some papers [8], [13], [18]–[20] that realize on-chip learning with FXP representation, but it still requires almost 16-bit for scratch training [21], and low-bit-precision (≤8-bit) training is only possible in the fine-tuning scenarios. Since the FXP-based DNN training required higher bit-precision compared with the FP representation, most on-chip learning processors adopted FP-based computing. It makes the on-chip learning processors unable to utilize the architecture of the conventional FXP-based energy-efficient DNN inference processors, such as lookup table [3] or bit-serial computing [5].

Currently, FXP-based low-bit-precision training methods are developed based on stochastic rounding (SR) [21]. The SR reduces rounding errors with probabilistic behavior. In other words, it can increase virtual representation resolution according to probabilistic rounding. In addition, the primal weight, the original weight before bit truncation, should be maintained as high bit-precision. The bit truncation only occurs during the FF and EP stages, but WG is performed with high bit-precision to update primal weight. Both DoReFa-NET [22] and Fxp-Net [23] successfully demonstrated DNN training with their custom DNNs, but they lost generality because the network modification was necessary for both methods.

III. Proposed Low-Bit-Precision Training Method

We introduced conventional low-bit-precision training methodologies, but they still required more than 8-bit to train general DNNs from the scratch. The proposed processor adopts FXP-based new number representation for the low-bit-precision training. Moreover, the HNPU adopts an active precision searching algorithm, which is the first trial to find optimal precision during the DNN training.

A. Stochastic Thresholding and Stochastic Dynamic Fixed-Point Representation

The HNPU adopts the previous low-bit-precision training scheme, SR, and primal weight. Equation (1) represents the SR operation

$$\mathrm{SR}(x) = \begin{cases} \lfloor x \rfloor + 1 & \text{with probability } x - \lfloor x \rfloor \\ \lfloor x \rfloor & \text{with probability } 1 - (x - \lfloor x \rfloor). \end{cases} \tag{1}$$

Although the SR increases virtual representation resolution, the numeric range is still limited due to low bit-width. The DFXP [2] representation resolves this problem by controlling its fraction length according to layer-wise distributions. In the DFXP-based computing, it counts overflow and decreases fraction length if the overflow appears. However, the distribution of the IA and weight is zero-centric, and only a minor portion of data has a large value and placed in the outlier of the distribution. Thus, many MSB bits are wasted to represent most small data. As shown in Fig. 2(a), strict fraction length control can precisely represent outlier data, but it can lose the DNN training accuracy because of inaccurate data representation that appeared in the majority of small data. To overcome this problem, fraction length can also be controlled only when the number of overflows is over the pre-determined threshold, but it can also be another reason for accuracy degradation because the outlier values can significantly affect the convolution results.

Algorithm 1 ST

Input: input vector $\boldsymbol{x}$, current integer length $\boldsymbol{i_t}$, bit-width $\boldsymbol{B}$, pre-determined overflow threshold $\boldsymbol{Th}$
Output: integer length of next iteration, $\boldsymbol{i_{t+1}}$

```
Stochastic threshold, Th_s = Th * Uniform(0, 1)
if Overflow_rate(x, i_t, B) ≥ Th_s then
    i_{t+1} = i_t + 1
else if Overflow_rate(x, i_t − 1, B) < Th_s then
    i_{t+1} = i_t − 1
else
    i_{t+1} = i_t
end if
```

As shown in Algorithm 1, the HNPU solves this problem by introducing stochastic thresholding (ST). The ST randomly determines the threshold value to control fraction length. If the overflow occurs, the convolution results are clipped to the nearest value current bit-precision can represent. The ST can enlarge the numeric representation range while minimizing the errors caused by clipping compared with the DFXP. Specifically, ST can compensate for the accuracy degradation when the training is done with less than 6 bit. Finally, a new number representation, SDFXP, is proposed by combining three different concepts, DFXP, SR, and ST (i: integer length and f: fraction length)

$$\mathrm{SDFXP}(x, i, f) = \begin{cases} 2^i - 2^{-f} & x \geq 2^i - 2^{-f} \\ 2^{-f} - 2^i & x \leq 2^{-f} - 2^i \\ 2^{-f} * \mathrm{SR}(2^f x) & \text{otherwise.} \end{cases} \tag{2}$$

As shown in Fig. 2(b), we compared the ResNet-9 training performance of DFXP and SDFXP in four different data sets. It shows that the required bit-precision is reduced from 10 to 4 bit in the MNIST and from 14 to 8 bit in the ImageNet data set compared with the DFXP-based training.

B. Layer-Wise Adaptive Precision Scaling

According to Fig. 2(b), the SDFXP successfully reduces the required bit-precision for DNN training. Furthermore, it shows that the required bit-precision can be varied according to the target data set even it utilizes the same network architecture. Despite this characteristic, conventional on-chip learning processors [8], [10]–[14], [16], [18]–[20] used fixed high bit-precision to train the DNNs because optimal precision searching has been considered impossible during the DNN training phase. Therefore, active bit-precision searching during the training can be the key functionality for the energy-efficient design of the on-chip learning processor.

The proposed processor supports automatic bit-precision searching by adopting a new training method, LAPS. Fig. 3 and Algorithm 2 show the outline of the LAPS

TABLE I
CNN TRAINING RESULT WITH SDFXP AND LAPS

Dataset	Network	FP16		FP8		SDFXP12		SDFXP8		SDFXP4		LAPS			
		Train Acc.	Test Acc.	Train Acc.	Test Acc.	Train Acc.	Test Acc.	Train Acc.	Test Acc.	Train Acc.	Test Acc.	Train Acc.	Test Acc.	Avg. Prec.	Weighted Avg. Prec.
MNIST	ResNet-9	99.90	99.22	99.28	98.78	99.82	99.15	99.84	99.21	99.32	99.01	99.40	99.02	**4.32**	**4.41**
	ResNet-18	99.73	99.18	99.60	99.22	99.71	99.17	99.75	99.21	99.30	99.05	99.14	98.88	**4.10**	**4.24**
	ResNet-50	99.56	98.82	99.44	99.08	99.66	99.12	99.57	99.25	99.08	99.04	99.24	99.03	**4.20**	**4.38**
	VGG-16	99.65	99.06	99.58	98.99	99.67	99.23	99.64	99.07	99.59	98.32	99.04	98.70	**4.00**	**4.00**
	SENet	99.75	99.20	99.63	99.10	99.68	99.06	99.70	99.13	99.20	98.78	99.22	98.83	**4.51**	**4.60**
CIFAR 10	ResNet-9	99.86	93.69	84.27	83.54	99.90	93.87	99.86	93.73	90.12	86.58	98.58	92.94	**5.84**	**6.27**
	ResNet-18	99.98	95.22	99.94	93.23	99.98	95.20	99.98	95.01	90.44	85.68	97.88	92.48	**5.14**	**5.53**
	ResNet-50	99.97	94.64	99.91	93.84	99.96	95.18	99.98	95.17	88.53	81.34	99.88	94.28	**6.02**	**8.00**
	VGG-16	99.95	93.82	99.90	92.86	99.94	93.41	99.93	93.69	89.22	86.02	99.78	93.11	**6.25**	**6.61**
	SENet	99.98	95.02	99.93	93.71	99.97	94.86	99.96	94.74	79.33	75.97	96.76	92.40	**7.07**	**8.00**
CIFAR 100	ResNet-9	98.17	73.14	26.83	21.08	98.82	73.48	98.64	73.27	65.27	57.51	97.57	72.31	**6.53**	**7.07**
	ResNet-18	99.96	77.33	99.92	73.44	99.97	78.37	99.96	78.18	83.74	62.85	98.14	75.02	**5.70**	**6.15**
	ResNet-50	99.92	78.04	64.47	48.82	99.96	79.01	99.95	79.28	78.74	60.44	99.35	74.22	**6.10**	**8.00**
	VGG-16	99.72	73.90	99.67	72.25	99.84	73.61	99.75	73.35	68.78	59.15	99.20	71.81	**6.54**	**6.89**
	SENet	99.94	77.33	99.90	75.72	99.96	76.71	99.95	76.63	79.58	62.31	99.65	75.16	**7.24**	**8.00**

TABLE II
IMAGENET (32 × 32 RESOLUTION) TRAINING RESULT WITH SDFXP AND LAPS

Dataset	Network	FP32		SDFXP12		SDFXP8		SDFXP4		LAPS			
		Train Acc.	Test Acc.	Train Acc.	Test Acc.	Train Acc.	Test Acc.	Train Acc.	Test Acc.	Train Acc.	Test Acc.	Avg. Prec.	Weighted Avg. Prec.
ImageNet 32×32	ResNet-9	59.02	47.79	58.90	47.23	57.07	46.62	31.94	26.35	56.90	46.47	**7.90**	**7.95**
	ResNet-18	84.41	53.52	82.96	53.44	62.45	50.40	41.21	30.82	79.31	53.51	**11.13**	**11.74**

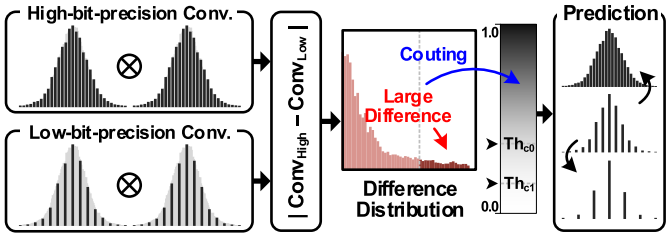


Fig. 3. Outline of LAPS.

algorithm. The LAPS calculates the difference between low-bit-precision and high-bit-precision convolution results to determine whether the DNN requires higher or lower bit-precision. The difference calculation occurs only in the first three iterations per epoch. It increases the counting numbers if the difference is larger than the predetermined threshold value. Then, the internal finite-state-machine increases or decreases the bit-precision of the corresponding layer based on the counting results. Determined precision is applied from the next iteration and it is maintained until an epoch is finished. In summary, the LAPS indirectly decides layer-wise optimal precision by comparing the low-bit-precision and high-bit-precision convolution results, and the precision is changed once for each epoch.

Table I shows the convolutional neural network (CNN) training results by using LAPS. We measured both the train and the test accuracy of four different DNNs in three typical data sets. It uses 16-bit primal weight both in FP- and SDFXP-based DNN training. The SDFXP8-based training shows similar or slightly better accuracy compared with the FP16 training. However, the SDFXP4-based training sometimes fails to achieve original high accuracy. The LAPS automatically finds the optimal precision of each layer to minimize the required bit-precision during the DNN training. One key observation is that the required bit-precision should be larger if the task becomes more difficult. In the MNIST training, the required precision is almost 4 bit, but it requires more than 6 bit during the CIFAR-100 training.

Table II shows additional ImageNet (32 × 32 resolution) training results with the SDFXP and LAPS. Although Table I simulation is done with 16-bit primal weight, it adopts 20-bit primal weight in ImageNet training. Consequently, the LAPS-based training automatically finds the layer-wise optimal precision even in various data sets and network types.

IV. OVERALL ARCHITECTURE

We introduced a new number representation, SDFXP, and an active precision searching methodology, LAPS. The proposed processor, HNPU, adopts both SDFXP and LAPS for high-speed and energy-efficient DNN training.

Fig. 4 shows the overall architecture of the HNPU, which consists of 32 bit-slice training cores (BSTCs), a peripheral training assistant, two optimizer cores (OPTCs), and a TOP RISC controller. Each BSTC has four processing element (PE)

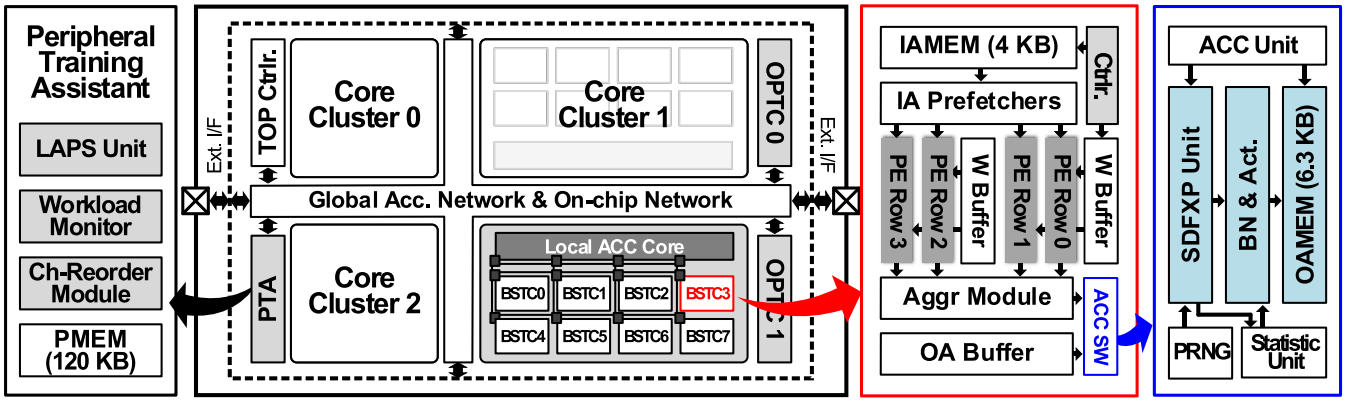


Fig. 4. Overall architecture of HNPU.

Algorithm 2 LAPS

Input: total iteration number of an epoch T, current iteration $t \in [0, T-1]$, input vector x, weight w, corresponding bit-width $B_{x,t}B_{w,t}$, difference threshold Th_d, count threshold Th_{c0}, Th_{c1}

Output: bit-width of the next iteration $B_{x,t+1}$, $B_{w,t+1}$

$C_{Low} = Conv(x, w, B_{x,t}, B_{w,t})$
if $t < 3$ **then**
  if $t = 0$ **then**
  $C_{High} = Conv(x, w, B_{x,t} + 2, B_{w,t})$
    if $Count(|C_{High} - C_{Low}| > Th_d) > Th_{c0}$ **then**
      $B_{x,t+1} = B_{x,t} + 1$
    else if $Count(|C_{High} - C_{Low}| > Th_d) < Th_{c1}$ **then**
      $B_{x,t+1} = B_{x,t} - 1$
    end if
  else if $t = 1$ **then**
  $C_{High} = Conv(x, w, B_{x,t}, B_{w,t} + 2)$
    if $Count(|C_{High} - C_{Low}| > Th_d) > Th_{c0}$ **then**
      $B_{w,t+1} = B_{w,t} + 1$
    else if $Count(|C_{High} - C_{Low}| > Th_d) < Th_{c1}$ **then**
      $B_{w,t+1} = B_{w,t} - 1$
    end if
  else if $t = 2$ **then**
  $C_{High} = Conv(x, w, B_{x,t} + 2, B_{w,t} + 2)$
    if $Count(|C_{High} - C_{Low}| > Th_d) > Th_{c0}$ **then**
      $B_{x,t+1} = B_{x,t} + 1$
      $B_{w,t+1} = B_{w,t} + 1$
    else if $Count(|C_{High} - C_{Low}| > Th_d) < Th_{c1}$ **then**
      $B_{x,t+1} = B_{x,t} - 1$
      $B_{w,t+1} = B_{w,t} - 1$
    end if
  end if
end if

rows, IA pre-fetchers, 4-kB IA memory, and buffers to store the weights and OAs. The entire memory included in the BSTC adopts double buffering to maintain high core utilization. The IA pre-fetcher includes four-entry FIFOs that temporally store input data without zero-slices (ZSs) and only non-zero-slices (NZSs) are fetched to the PE rows. Accumulation is performed inside the PE rows. The aggregation module collects the accumulation results to complete convolution or matrix multiplication operations. Final results are transferred to the OA buffer or accumulation switch (ACC SW). The ACC SW includes an SDFXP unit that performs post-processing to convolution results. It has the role of batch normalization (BN) and activation function computing, which is an essential part of DNN training. Every BSTC is connected with both the data NoC and the 2-D mesh-type AB-RAN.

The peripheral training assistant consists of an LAPS unit, a workload monitor, and a channel reordering module with a 120-kB peripheral memory (PMEM). It is used as the temporal buffer to store the IAs or weights just before and after external memory access. It simultaneously monitors sparsity patterns of IA and reflects it for workload balancing. Moreover, the channel reordering unit performs a weight transpose operation during EP computing. The LAPS unit compares the low-bit-precision and high-bit-precision convolution results that are temporarily stored in the PMEM. The last unit, OPTC, can support three types of optimizers: SGD, momentum, and signADAM++ [24].

V. Proposed Processor: HNPU

Section III introduced algorithm-level key features adopted in the HNPU. Architecture- and circuit-level features will be described in this section. First, the bit-slice serial architecture with the pseudorandom-number-based SDFXP unit and LAPS unit will be explained. An explanation of energy-efficient IOSS core architecture together with AB-RAN will follow.

A. Bit-Slice Serial Architecture With SDFXP and LAPS Unit

1) Bit-Slice Training Unit With Bit-Slice Serial Computing: Fig. 5 shows the detailed computation flows of the HNPU. The main DNN core, BSTC, includes a bit-slice training unit and an SDFXP unit. The bit-slice training unit uses 4 bit × 4 bit MAC as a base unit and receives a four bit-slices at a time after dividing high-bit-precision data into a number of

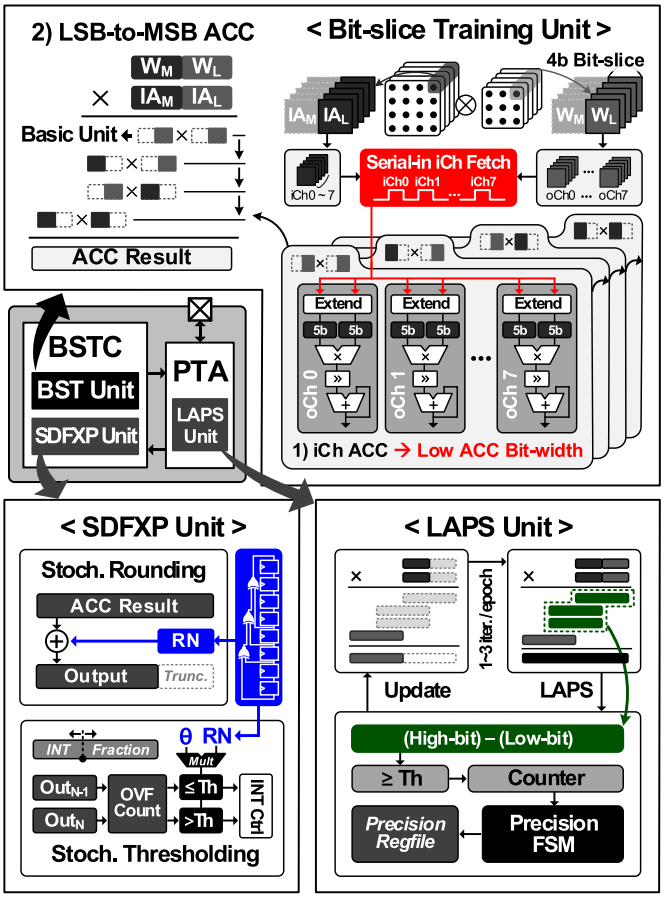


Fig. 5. BSTC with SDFXP and LAPS unit.

four bit-slices. Conventional bit-serial architecture [5] used in many bit-reconfigurable processors requires high accumulation bit-width to prevent swamping problem [17]. The main reason is that it computes LSB-to-MSB accumulation before it calculates further operands. To solve this problem, the HNPU computes input channel accumulation before the move to the MSB data. Since it first accumulates the LSB data, it prevents swamping problem even with the small bit-width. As a result, the proposed bit-slice serial architecture reduces accumulation bit-width by 38.1% due to the change of accumulation order.

2) SDFXP Unit With Pseudorandom Number Generator: After the bit-slice training unit completes the convolution, final accumulation results are transferred to the SDFXP unit. The SDFXP unit includes a linear-feedback-shift-register-based pseudorandom number generator and the random number (RN) is used for the SR and the ST. The SDFXP unit can support three different rounding modes: 1) no-rounding; 2) normal-rounding; and 3) SR. Based on training methodologies, one of the rounding modes is selected. In our CNN training setup, the SR is used during the FF and EP stage, but the no-rounding mode is adopted during the WG stage. After the SR, the ST is performed and the updated integer length is used from the next iteration.

3) LAPS Unit: As explained in Section III, the HNPU adopts LAPS for active precisions searching. The proposed processor calculates both low-bit-precision and high-bit-precision convolution results. During the 4-bit computing, the BSTC can minimize throughput degradation by considering partial multiplication results directly as the computing difference. Since the difference calculation is done only in a few iterations, the LAPS-based precision search does not affect the entire training speed. The BSTC can respond to the bit-precision changes and achieves faster training compared with the fixed bit-width DNN training [8], [10]–[14], [16], [18]–[20].

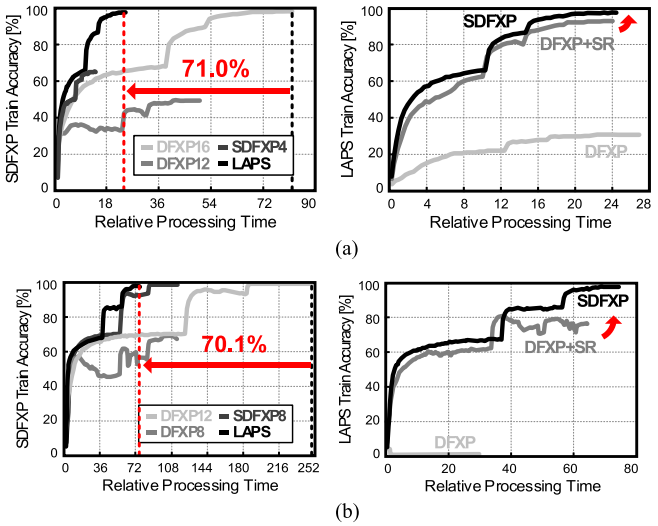


Fig. 6. CNN training result with the proposed processor. (a) ResNet-9 training result in the CIFAR-100 data set. (b) ResNet-18 training result in the CIFAR-100 data set.

4) Training Performance Improvement Result With Bit-Slice Serial Architecture: Fig. 6 shows the training results with the BSTC, including SDFXP and LAPS unit. Previously, training of ResNet-9 and ResNet-18 requires more than 12 bit when it adopts DFXP. It spends 9× longer time compared with the 4-bit training. When the HNPU adopts SDFXP representation, the required bit-precision is reduced to 8-bit and induces 55.6% processing time reduction. It can further increase speed and efficiency by adopting LAPS and it finally shows 3.4× speedup and 5.6× higher energy efficiency compared with the DFXP-based training.

B. IOSS Core Architecture

Previous on-chip learning processors [10]–[13] tried to utilize word-level sparsity to improve both throughput and energy efficiency. Therefore, their performance highly depended on the sparsity caused by ReLU. Input sparsity exploitation such as LNPU is not efficient for DNN training because the EP does not induce input sparsity. Dual-sparsity-aware design can overcome this problem, but it cannot utilize both input and output sparsities simultaneously because only one of the sparsity types, either input or output sparsity, appears during the training. Output zero prediction can be a solution to fully utilize both input and output sparsities at the same time, but it causes area and energy overhead when the prediction block becomes huge to achieve fast prediction.

1) Slice-Level Input Skipping: As shown in Fig. 7, the HNPU releases ReLU dependence by utilizing slice-level

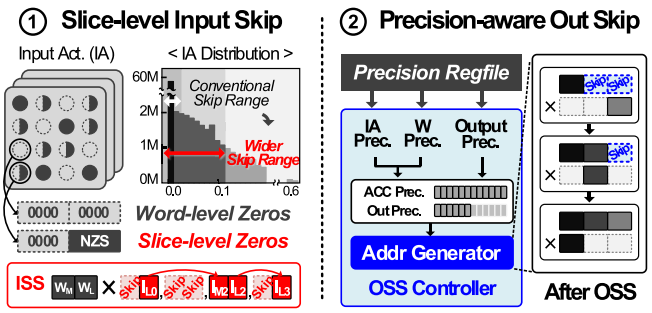


Fig. 7. Outline of ISS and OSS.

sparsity. The DNN usually shows the Gaussian-like distribution in IAs after the BN layer. Among them, half of the IA is evaluated as zero because of the ReLU. In addition, the majority of data are concentrated near zero and they are significantly smaller than the maximum values of the distribution. In that case, near-zero data waste MSB bits when using the FXP representation. It gives great potential to the slice-level sparsity exploitation. Since the slice-level input sparsity does not depend on ReLU, it can also appear during the EP stage. The HNPU can utilize more than 60% input-slice sparsity both in FF and EP stages. Unlike the conventional dual-sparsity-aware hardware [11]–[13], the HNPU can achieve high throughput during the EP stage even without the output sparsity exploitation unit.

2) Precision-Aware Output Skipping: Unlike FP-based computing, FXP-based computing induces high multiplication bit-width. On the contrary, the required bit-precision of the next layer is similar to its operands. This characteristic makes negligible accumulations during BSS computing. For example, 12 bit $\times$ 12 bit multiplication requires 24 bit for output representation and it requires a larger bit-width to cover multi-channel accumulation without truncation. Even though the 12 bit $\times$ 12 bit convolution induces high accumulation bit-width, the required bit-width for the next layer is as similar as 12 bit. The accumulation results should be truncated to 12 bit, and the LSB accumulation information does not affect the final results. Precision-aware output skipping finds out these negligible computations in the LSB accumulation slices and excludes them before it fetches them as the operands. In the abovementioned example, 12 bit $\times$ 12 bit convolution is divided into nine accumulation sequences and three LSB accumulation slices are excluded in the OSS computing. The throughput improvement appears only when the required precision is more than 4-bit because 4 bit $\times$ 4 bit convolution consists of only a single accumulation slice.

If the processor directly uses the FP32-based pre-trained model for inference, the accuracy can be degraded due to the OSS. Network fine-tuning with the OSS scheme can compensate for the performance degradation. In addition, the DNN training performance is not degraded if the OSS is applied from the scratch training. Table III shows how the OSS affects the DNN training performance. Since we adopt SR and it adds RNs to the LSB accumulation results, the importance of the LSB accumulation values is minor and it does not degrade its training performance.

TABLE III
ON-CHIP LEARNING PROCESSOR PERFORMANCE COMPARISON TABLE

Network	ResNet-9				ResNet-18			
Precision	SDFXP8		SDFXP12		SDFXP8		SDFXP12	
OSS	X	O	X	O	X	O	X	O
Train Acc.	98.64	97.96	98.82	98.73	99.96	99.96	99.97	99.97
Test Acc.	73.27	73.36	73.48	73.77	78.18	77.34	78.37	77.48

3) IOSS Core Architecture: IOSS core architecture can be divided into two major parts. The first part is the IA pre-fetcher, which performs ISS by excluding ZS. Each PE row has an IA pre-fetcher and a total of four IA pre-fetchers share an input activation memory (IAMEM). The HNPU uses IA pre-fetchers to detect the ZS, and the internal ZS counter generates zero-slice-length (ZSL) that is used as the address of the weight buffer. Only the remaining NZS is stacked in an FIFO and then fed into the PE arrays (PAs) sequentially. When the NZS is fetched from FIFO, it is broadcasted to compute eight different output channels. Since the different PAs have the role of different input channel computing, the accumulation results are transferred to the aggregation module to complete the convolution operation. The ISS, together with bit-slice serial architecture, shows a 7.9$\times$ higher throughput at 90% slice-sparsity in 8-bit training cases.

The other major part is the OSS controller that omits useless accumulations based on the precision information of the current and next layers stored in the precision register file. The main role of the OSS controller is generating sequential BSS-related instructions. When the OSS controller judges to skip partial accumulations, it skips corresponding instructions before it affects main core computing. It additionally manages accumulation bit-width by using the bit-shift operation. Moreover, it modifies the address of the weight buffer and OAMEM simultaneously. With the OSS, the throughputs of 8-, 12-, and 16-bit convolution are improved by 33%, 50%, and 67%, respectively.

As shown in Fig. 8, combining ISS and OSS, the IOSS architecture compensates for the throughput degradation that occurred in high-bit-precision computing and it increases the normalized throughput and energy efficiency by 59.9% and 82.9% compared with the GANPU. One of the main reasons for the improvement is that slice-level skipping induces more sparsity compared with word-level skipping. The other reason is that it can find out negligible operations appeared in IA and OA and utilize both without any output prediction unit.

C. AB-RAN for Variable Output Bandwidth Supporting

The GANPU can support multi-DNN allocation by adopting a reconfigurable accumulation network (RAN), which increases the degree of freedom for the accumulation path. Due to the RAN, it can successfully balance computation- and memory-bound computing. Despite its reconfigurable functionality, conventional RAN architecture is based on FP representation and it can support only single precision. It is not suitable for the HNPU because it shows different throughputs according to the variable bit-width determined by the LAPS algorithm. The HNPU, therefore, adopts the basic concept

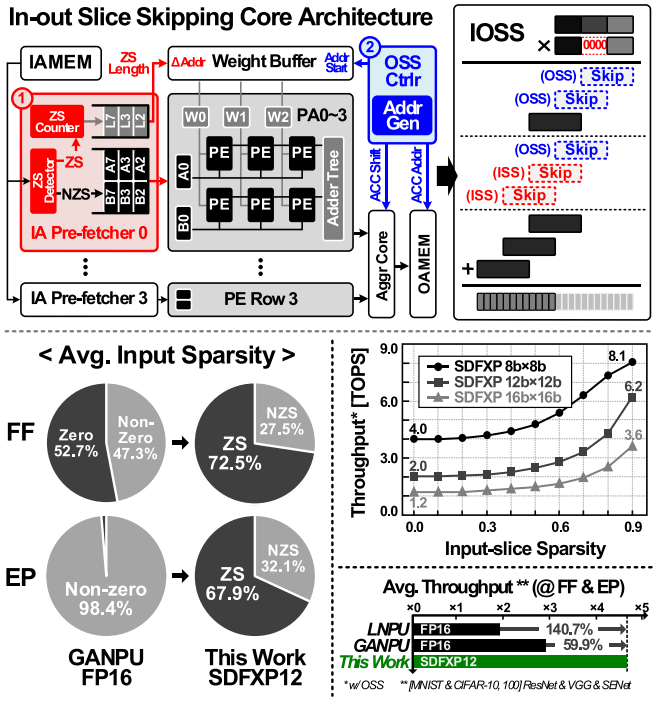


Fig. 8. IOSS core architecture and throughput improvement result (comparison with LNPU and GANPU was performed in the same 65-nm CMOS technology).

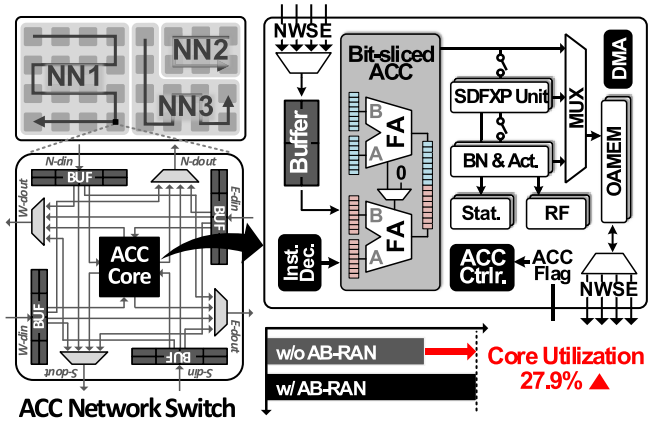
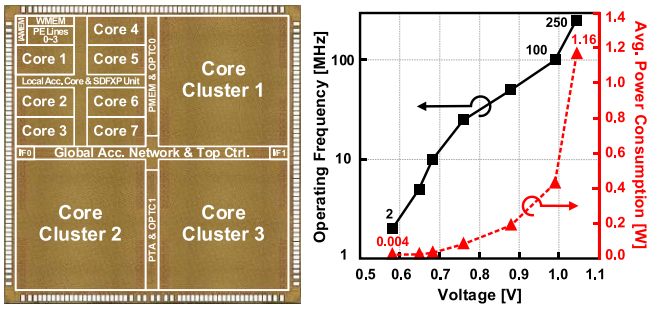


Fig. 9. Architecture of bit-sliced adder-based RAN: AB-RAN.

of RAN to support multi-DNN allocation but revises it as AB-RAN for variable output bandwidth supporting.

As shown in Fig. 9, the main difference of the AB-RAN is bit-sliced adder-based accumulation. Bit-sliced accumulation can support variable output bandwidth by dividing or combining multiple adders. The bit-sliced adder of the AB-RAN can process wide-range output bandwidth with the minimum number of adders and bit-width. Compared with the same number of high-bit-precision adders, the bit-slice adder-based accumulation can increase the core utilization by 27.9% even with the bit-width variation from 4 to 16 bit. Furthermore, the AB-RAN includes a statistic unit that sequentially calculates the mean and variance of the convolution results. The mean and variance are also calculated by using bit-sliced



	Specifications					
Technology	Samsung 28nm 1P8M CMOS					
Die Area	3.6mm × 3.6mm (12.96mm^2)					
On-chip SRAM	552 KB					
Op. Condition	0.58V (@ 2MHz) ~ 1.04V (@ 250MHz)					
Data Type	DFXP or SDFXP (4/8/12/16)-bit × (4/8/12/16)-bit					
Area Efficiency	59-to-948 GOPS/mm^2					
	In-slice Sparsity		0%		90%	
Power [mW]	250MHz, 1.04V		1162		500	
	2MHz, 0.58V		4.39		1.95	
			w/o OSS	w/ OSS	w/o OSS	w/ OSS
Peak Performance [TOPS]	250MHz	16b×16b	0.77	1.23	2.27	3.63
		12b×12b	1.37	2.05	4.15	6.22
		8b×8b	3.07	4.10	6.09	8.13
		4b×4b	12.3			
Energy Efficiency [TOPS/W]	250MHz	16b×16b	0.66	1.06	4.53	7.25
		12b×12b	1.18	1.76	8.29	12.4
		8b×8b	2.64	3.53	12.2	16.2
		4b×4b	10.6		24.6	
	2MHz	16b×16b	1.40	2.24	9.29	14.9
		12b×12b	2.49	3.73	17.0	25.5
		8b×8b	5.60	7.46	25.0	33.3
		4b×4b	22.4		50.3	

Fig. 10. Chip photograph and performance summary.

adders. The BN unit sequentially calculates mean and variance as soon as the main core generates the outputs, so it can compute BN layers without the main core throughput degradation.

VI. Measurement Results

A. Chip Implementation Results

Fig. 10 shows the chip implementation result and the performance summary. The HNPU is fabricated with the SAMSUNG 28-nm CMOS technology and shows 3.6 mm × 3.6 mm area occupation. The operating condition is varied from 2- to 250-MHz clock frequency with the 0.58-to-1.04-V supply voltage. Even in the same operating condition, power consumption can also be varied according to input slice sparsity. In the maximum frequency condition, the HNPU shows 500-to-1162-mW power consumption according to slice-level sparsity. The maximum energy efficiency appears when the clock frequency is 2 MHz, and the corresponding power consumption appears as 1.95-to-4.39 mW. Since the ratio of zero-bit-slice is increased in the PA, it shows lower dynamic power consumption in the higher sparsity ratio.

The throughput of the HNPU can be varied because of the three factors; 1) bit-precision; 2) in-slice sparsity; and 3) OSS. The peak performance always appears at the 4 bit × 4 bit computation, but the corresponding throughput is not varied

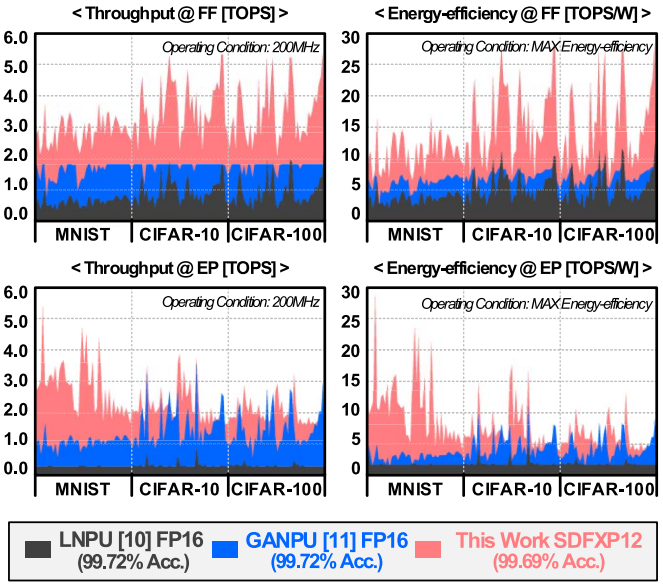


Fig. 11. Layer-wise throughput and energy-efficiency comparison result without LAPS. It includes the training results of four DNNs (ResNet-9, ResNet-18, VGG-16, and SENet) and three data sets (MNIST, CIFAR-10, and CIFAR-100).

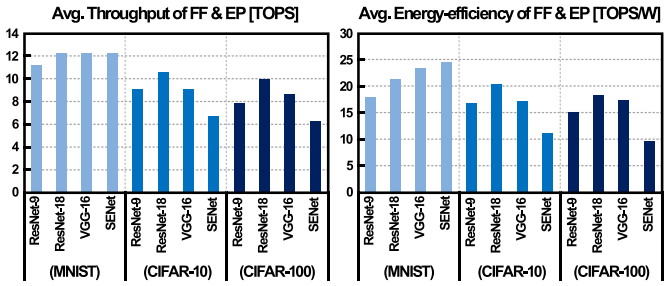


Fig. 12. Data set-wise throughput and energy-efficiency comparison result with LAPS (measurement result of the last epoch including both the first and last layer results).

according to in-slice sparsity and OSS option. On the other hand, the throughput during high-bit-precision computing (>4 bit) is affected by slice-level sparsity and OSS option. When the input slice has 90% sparsity and utilizes OSS, the throughput of the 8-to-16-bit computing shows 2.65-to-4.71× higher than the baseline. The efficiency of the high-bit-precision training is also affected by its precision and input slice sparsity so that it varies from 1.4 TOPS/W (16-bit without IOSS) to 33.3 TOPS/W (8-bit with IOSS). Although there is no ZS-skipping during 4-bit computing, the power consumption is reduced due to the zero gating. The peak energy efficiency appears as 50.3 TOPS/W at 90% slice-level sparsity with 4 bit × 4 bit computing.

B. CNN Training Benchmark Results

Many previous papers emphasized just peak performance and energy efficiency by using high sparsity and low-bit-precision computing conditions. However, it is impractical because most DNNs do not be placed in those computing conditions. Specifically, the consideration of low-bit-precision computing should be cautious because it can induce accuracy degradation. For the fair comparison, we composed a CNN training benchmark with three different image classification data sets and widely used networks, such as ResNet-9, ResNet-18, VGG-16, and SENet. The measurement results of the first and the last layer are excluded in Figs. 11 and 13 and Table IV because the sparsity pattern of the first and the last layer differs from the other layers.

In the EP stage, the performance of the LNPU [10] is much lower than that of the GANPU [11] and the HNPU because there is no word-level input sparsity. In contrast, the GANPU can utilize output sparsity, but it still cannot utilize input sparsity. Since the HNPU exploits slice-level input sparsity, the HNPU shows higher throughput and energy efficiency even without word-level output zero skipping.

In the FF stage, the HNPU still shows the highest performance compared with the other on-chip learning processors. Although the GANPU can utilize dual sparsity by output zero prediction, the throughput of the main core is restricted because it should wait until the prediction is completed. The HNPU minimizes its energy consumption by excluding output zero prediction and deterministic OSS successfully compensates for the efficiency degradation.

We summarized the performance improvement results and benchmark comparison results in Figs. 11–13. Conventional FXP-based training requires ≥16 bit to achieve high accuracy regardless of network type or data set. It becomes the main obstacle to energy-efficient DNN training. The HNPU adopts a new number representation, SDFXP, and improves the training energy efficiency by 4.5×. In addition, IOSS and AB-RAN further increase their energy efficiency by 2.4×. Finally, the LAPS enhances the energy efficiency more by 1.5×. The LAPS algorithm fixes bit-precision of the first and last layer as 12-bit and determines optimal bit-precision of the other intermediate layers every iteration. Based on the benchmark result, the proposed processor shows at least 5.9× higher energy efficiency and 2.5× higher area efficiency compared with the previous on-chip learning processors [10], [11], [16], [18]–[20], [25].

Table V shows the energy-efficiency comparison results during the ImageNet (32 × 32 resolution) training. Since the LNPU [8] can support FP16/FP8 mixed precision, we adopt a 0.1/0.9 ratio of FP16/FP8 for ImageNet training. Although GANPU [9] cannot support mixed precision, it can utilize word-level output sparsity during the EP stage. Moreover, it can support dual sparsity during the FF stage. The proposed processor, HNPU, shows the highest energy efficiency due to the slice-level input sparsity exploitation and layer-wise bit-precision optimization with a new number representation, SDFXP.

As analyzed in previous work [19], the WG stage can be considered as the computing of a fully connected layer. This characteristic disturbs spatial data reuse that appeared in the convolutional layer so that it induces 33.3% core utilization in the HNPU. Furthermore, the proposed processor adopts high-bit-precision primal weight, so it can utilize ISS but OSS cannot be applied. The throughput improvement through the ISS can also be limited because the main reason for the throughput bottleneck in the WG stage is limited external memory bandwidth.

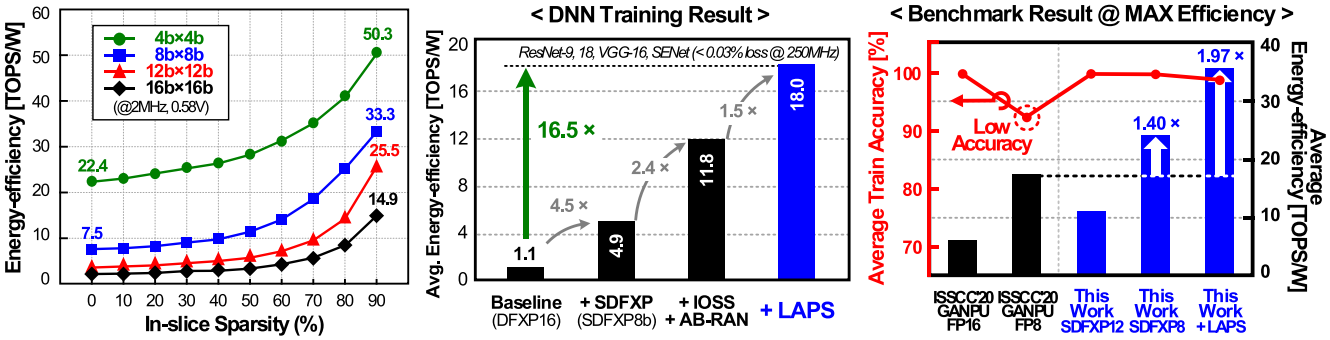


Fig. 13. HNPU measurement results. (a) Energy efficiency according to in-slice sparsity ratio. (b) Efficiency improvement result with the proposed schemes. (c) Training accuracy and efficiency comparison with GANPU (based on the benchmark explained in Section VI-B).

TABLE IV

ON-CHIP LEARNING PROCESSOR PERFORMANCE COMPARISON TABLE (BASED ON THE BENCHMARK EXPLAINED IN SECTION VI-B)

			V100 [25]	S. VLSI 19 [18]	ISSCC 19 [10]	SSCL 20 [20]	JSSC 20 [19]	ISSCC 20 [11]	S. VLSI 20 [16]	This Work
Skipping Technique			X	X	IA skip	X	X	IA & OA skip	X	IS & OS skip
Process [nm]			12	65	65	65	65	65	14	28
Die Area [mm^2]			815	5.76	16.0	16.9	10.24	32.4	9.8	12.96
Supply Voltage [V]			-	0.78-1.1	0.78-1.1	0.55-1.0	0.65-1.0	0.70-1.1	0.54-0.80	0.58-1.04
Max Frequency [MHz]			1530	200	200	294.4	160	200	1500	250
Precision		W	FP 16/32/64	FXP16	FP 8/16	FXP16	DFXP8	FP 8/16	FP 16/32	SDFXP 4/8/12/16
		IA	FP 16/32/64	FXP13	FP 8/16	FXP16	DFXP8	FP 8/16	FP 16/32	SDFXP 4/8/12/16
Benchmark Average Performance	Avg. Energy Efficiency @ MAX Efficiency Condition[1] [TOPS/W or TFLOPS/W]	FF	0.42 (FP16)	0.70 (FXP16)	5.11 (FP16)	2.60 (FXP16)	1.03 (DFXP8)	6.93 (FP16)	1.40 (FP16)	**34.06 (SDFXP+LAPS)**
		EP	0.42 (FP16)	0.35 (FXP16)	1.86 (FP16)	2.60 (FXP16)	1.03 (DFXP8)	5.22 (FP16)	1.40 (FP16)	**37.09 (SDFXP+LAPS)**
	Avg. Area Efficiency @ MAX Throughput Condition[1] [GOPS/mm^2 or GFLOPS/mm^2]	FF	153.4 (FP16)	17.78 (FXP16)	54.70 (FP16)	8.93 (FXP16)	13.10 (DFXP8)	54.01 (FP16)	306.1 (FP16)	**757.61 (SDFXP+LAPS)**
		EP	153.4 (FP16)	8.89 (FXP16)	19.89 (FP16)	8.93 (FXP16)	13.10 (DFXP8)	39.68 (FP16)	306.1 (FP16)	**731.67 (SDFXP+LAPS)**

1) Operating conditions in which the maximum efficiency or throughput of each processor appears

TABLE V

ENERGY EFFICIENCY (TOPS/W OR TFLOPS/W) DURING THE IMAGENET (32 × 32 RESOLUTION) TRAINING

Network	ResNet-9			ResNet-18		
Processor	LNPU [10]	GANPU [11]	HNPU	LNPU [10]	GANPU [11]	HNPU
FF Stage	2.41	7.03	14.46	4.27	7.14	15.16
EP Stage	4.68	5.49	5.69	8.20	5.61	6.17
Average	3.54	6.26	10.08	6.24	6.37	10.66

C. Online Learning-Based Mobile Network Optimization Demonstration Result

In Section VI-B, the HNPU shows energy-efficient DNN training results. As mentioned in Section I, the DNN training can give an opportunity to improve the performance of the mobile-oriented network in practical situations. Online learning can be a solution of accuracy compensation, but the labeling was the main restriction to utilize the training in practical scenarios. TKD [9] suggested self-labeling-based online learning and it can optimize both the throughput and the energy efficiency during the DNN inference. TKD adopts knowledge distillation, which gets labels from the large teacher network, and its inference results are used as the label to train the small student network.

We demonstrated our processor with the TKD-based online learning scenarios. The student network is mainly used, but its parameters are updated when the detection accuracy becomes low. In addition, it periodically distills the knowledge of the teacher network to adapt the environmental changes. The HNPU spends most of the time accelerating Tiny-YOLO-v3 inference but calculates both inferences of the YOLO-v3 and Tiny-YOLO-v3 every five frames. After that, the HNPU updates the parameters of the decoder layers by utilizing YOLO-v3 inference results and repeats this process for three iterations before receiving a new frame.

Fig. 14 shows the TKD-based object detection demonstration results using the HNPU. We utilized DDR4 SDRAM as the external memory to store both network parameters and intermediate data that occurred during the computations. The HNPU utilized SDFXP for a low-bit object detection demonstration. The precisions of the network were determined by the LAPS algorithm during the pre-training. Instead, the precision determined during the pre-training is maintained during online learning. Both YOLO-v3 and Tiny-YOLO-v3 finally required 8-bit precision for inference except the first

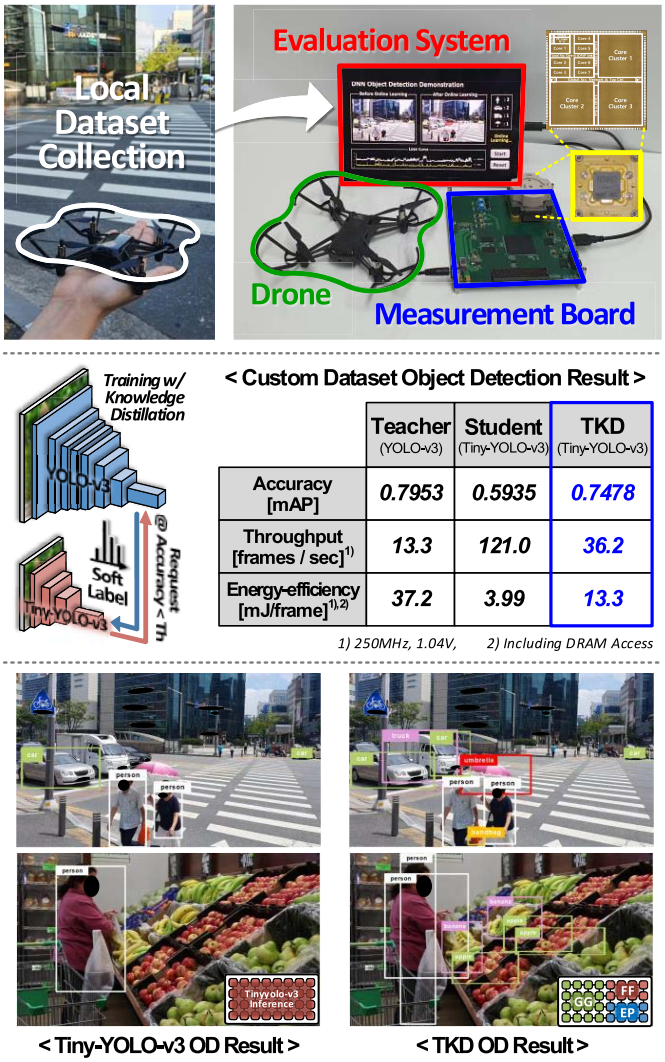


Fig. 14. Demonstration result: TKD-based energy-efficient object detection demonstration result.

and the last layer. These two layers used 16-bit precision. Furthermore, the precision requirements for EP and WG stages were determined as 16-bit.

In the YOLO-v3 acceleration scenario, it shows the highest object detection accuracy, mean average precision (mAP). However, the HNPU cannot achieve real-time conditions [>30 frame-per-second (FPS)] because of the heavy computation amounts required for YOLO-v3. The HNPU-based TKD acceleration shows an average of 36.2-FPS throughput while achieving 0.7479-mAP detection accuracy. Moreover, it can save 64.2% energy consumption compared with the YOLO-v3 acceleration. Due to the HNPU-based online learning, it successfully achieves high accuracy in object detection applications even in the use of the mobile-oriented network. In addition, the IOSS-based bit-slice serial architecture with SDFXP and LAPS algorithm minimizes DNN training overhead and finally accomplishes energy-efficient DNN applications in mobile devices.

VII. Conclusion

The HNPU, an ultra-low-bit-width (<8-bit) DNN training processor, is proposed for energy-efficient DNN training in mobile devices. Conventional DNN training processors only focused on improving throughput and energy efficiency by utilizing sparsity induced by the ReLU. However, the proposed processor lowers the dependence on ReLU, and instead, it automatically optimizes its required bit-precision during the DNN training. The proposed new number representation, SDFXP, induces 4.5× higher energy-efficiency compared with the conventional DFXP-based training. When the SDFXP representation is combined with the active precision searching method, LAPS, it can achieve 53.1% higher energy efficiency in the CNN training benchmark. Not only bit-precision but also slice-level sparsity is utilized to maximize both throughput and efficiency. Slice-level input skipping can exploit 1.38-to-42.4× more sparsity compared with the word-level zero-skipping core architectures. Furthermore, it skips useless accumulations according to layer-wise variable precision determined by LAPS. Combining ISS and OSS shows at least 5.9× higher energy efficiency compared with the previous state-of-the-art DNN training processors [10], [11], [16], [18]–[20], [25]. Finally, the new internal accumulation network, AB-RAN, helps HNPU to maintain its high throughput even in various bit-width conditions.

Due to the algorithm-hardware co-optimization, an adaptive DNN training processor with the SDFXP and LAPS as well as IOSS and AB-RAN accomplishes state-of-the-art energy efficiency, 50.3 TOPS/W. Moreover, the HNPU successfully demonstrates both CNN training from scratch and online-learning-based optimization.

The proposed LAPS algorithm shows the energy-efficient DNN training due to the automatic optimal precision search. However, the current LAPS algorithm induces lower bit-precision training, so it shows low training accuracy during the ImageNet training. In future work, we will advance LAPS to support both energy-efficient DNN training and accurate DNN training. Moreover, we will continue the research on minimizing the required bit-precision of primal weight.

References

[1] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE J. Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, Jan. 2017.

[2] D. Shin, J. Lee, J. Lee, and H.-J. Yoo, "14.2 DNPU: An 8.1TOPS/W reconfigurable CNN-RNN processor for general-purpose deep neural networks," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, San Francisco, CA, USA, Feb. 2017, pp. 240–241.

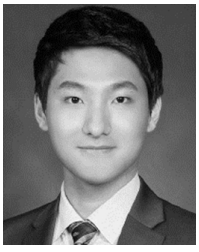
[3] J. Lee, D. Shin, and H.-J. Yoo, "A 21 mW low-power recurrent neural network accelerator with quantization tables for embedded deep learning applications," in *Proc. IEEE Asian Solid-State Circuits Conf. (A-SSCC)*, Seoul, South Korea, Nov. 2017, pp. 237–240.

[4] K. Ueyoshi *et al.*, "QUEST: A 7.49TOPS multi-purpose log-quantized DNN inference engine stacked on 96 MB 3D SRAM using inductive-coupling technology in 40 nm CMOS," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, San Francisco, CA, USA, Feb. 2018, pp. 216–218.

[5] J. Lee, C. Kim, S. Kang, D. Shin, S. Kim, and H.-J. Yoo, "UNPU: A 50.6TOPS/W unified deep neural network accelerator with 1b-to-16b fully-variable weight bit-precision," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, San Francisco, CA, USA, Feb. 2018, pp. 218–220.

[6] Z. Yuan *et al.*, "Sticker: A 0.41-62.1 TOPS/W 8Bit neural network processor with multi-sparsity compatible convolution arrays and online tuning acceleration for fully connected layers," in *Proc. IEEE Symp. VLSI Circuits*, Honolulu, HI, USA, Jun. 2018, pp. 33–34.

[7] J. Konečný *et al.*, "Federated learning: Strategies for improving communication efficiency," in *Proc. NIPS Workshop Private Multi-Party Mach. Learn.*, 2016, pp. 1–10.
[8] D. Han, J. Lee, J. Lee, and H.-J. Yoo, "A low-power deep neural network online learning processor for real-time object tracking application," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 66, no. 5, pp. 1794–1804, May 2019.
[9] M. Farhadi and Y. Yang, "TKD: Temporal knowledge distillation for active perception," in *Proc. IEEE Winter Conf. Appl. Comput. Vis. (WACV)*, Snowmass Village, CO, USA, Mar. 2020, pp. 942–951.
[10] J. Lee, J. Lee, D. Han, J. Lee, G. Park, and H.-J. Yoo, "7.7 LNPU: A 25.3TFLOPS/W sparse deep-neural-network learning processor with fine-grained mixed precision of FP8-FP16," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, San Francisco, CA, USA, Feb. 2019, pp. 142–144.
[11] S. Kang *et al.*, "7.4 GANPU: A 135TFLOPS/W multi-DNN training processor for GANs with speculative dual-sparsity exploitation," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, San Francisco, CA, USA, Feb. 2020, pp. 140–142.
[12] S. Kim, J. Lee, S. Kang, J. Lee, and H.-J. Yoo, "A 146.52 TOPS/W deep-neural-network learning processor with stochastic coarse-fine pruning and adaptive input/output/weight skipping," in *Proc. IEEE Symp. VLSI Circuits*, Honolulu, HI, USA, Jun. 2020, pp. 1–2.
[13] F. Tu *et al.*, "Evolver: A deep learning processor with on-device quantization–voltage–frequency tuning," *IEEE J. Solid-State Circuits*, vol. 56, no. 2, pp. 658–673, Feb. 2021.
[14] *Tensor Processing Unit—Second Generation (TPU-V2)*, Google, Menlo Park, CA, USA.
[15] U. Köster *et al.*, "Flexpoint: An adaptive numerical format for efficient training of deep neural networks," in *Proc. 31st Int. Conf. Neural Inf. Process. Syst. (NIPS)*. Red Hook, NY, USA: Curran Associates, 2017, pp. 1740–1750.
[16] J. Oh *et al.*, "A 3.0 TFLOPS 0.62V scalable processor core for high compute utilization AI training and inference," in *Proc. IEEE Symp. VLSI Circuits*, Honolulu, HI, USA, Jun. 2020, pp. 1–2.
[17] N. Wang *et al.*, "Training deep neural networks with 8-bit floating point numbers," in *Proc. 32nd Int. Conf. Neural Inf. Process. Syst. (NIPS)*. Red Hook, NY, USA: Curran Associates, 2018, pp. 7686–7695.
[18] D. Han, J. Lee, J. Lee, and H.-J. Yoo, "A 1.32 TOPS/W energy efficient deep neural network learning processor with direct feedback alignment based heterogeneous core architecture," in *Proc. Symp. VLSI Circuits*, Kyoto, Japan, Jun. 2019, pp. C304–C305.
[19] S. Choi, J. Sim, M. Kang, Y. Choi, H. Kim, and L.-S. Kim, "An energy-efficient deep convolutional neural network training accelerator for *in situ* personalization on smart devices," *IEEE J. Solid-State Circuits*, vol. 55, no. 10, pp. 2691–2702, Oct. 2020.
[20] S. Yin and J.-S. Seo, "A 2.6 TOPS/W 16-bit fixed-point convolutional neural network learning processor in 65-nm CMOS," *IEEE Solid-State Circuits Lett.*, vol. 3, pp. 13–16, 2020.
[21] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *Proc. 32nd Int. Conf. Int. Conf. Mach. Learn.*, vol. 37, 2015, pp. 1737–1746.
[22] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, "DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients," 2016, *arXiv:1606.06160*. [Online]. Available: http://arxiv.org/abs/1606.06160
[23] X. Chen, X. Hu, H. Zhou, and N. Xu, "FxpNet: Training a deep convolutional neural network in fixed-point representation," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Anchorage, AK, USA, May 2017, pp. 2494–2501.
[24] D. Wang *et al.*, "SignADAM++: Learning confidences for deep neural networks," in *Proc. Int. Conf. Data Mining Workshops (ICDMW)*, Beijing, China, Nov. 2019, pp. 186–195.
[25] NVIDIA. *Tesla V100*. [Online]. Available: https://www.nvidia.com/ko-kr/data-center/v100



Donghyeon Han (Graduate Student Member, IEEE) received the B.S. degree in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea, in 2017, where he is currently pursuing the M.S. degree.

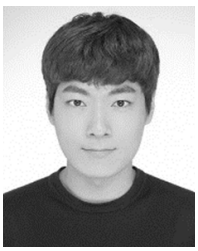
His current research interest includes low-power system-on-chip design, especially focused on deep neural network accelerators and hardware-friendly algorithms for deep learning.

Dongseok Im (Graduate Student Member, IEEE) received the B.S. degree in electrical engineering from the Pohang University of Science and Technology (POSTECH), Pohang, South Korea, in 2018, and the M.S. degree in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea, in 2020, where he is currently pursuing the Ph.D. degree.

His current research interests include energy-efficient deep learning system-on-chip (SoC) design and intelligent vision systems.

Gwangtae Park (Graduate Student Member, IEEE) received the B.S. degree in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea, in 2015, where he is currently pursuing the M.S. degree.

His current research interests include hardware/software co-design to implement low-power system on chip (SoC) for deep neural network inference/training.



Youngwoo Kim (Graduate Student Member, IEEE) received B.S. and M.S. degrees in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea, in 2018 and 2020, respectively.

His current research interests include low-power memory-efficient architecture and hardware-oriented algorithms, especially focused on deep learning system-on-chip.

Seokchan Song (Graduate Student Member, IEEE) received the B.S. degree in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea, in 2021, where he is currently pursuing the M.S. degree.

His current research interest includes low-power system-on-chip design, especially focused on mixed-mode deep neural network accelerators and hardware-friendly algorithms for deep learning.

Juhyoung Lee (Graduate Student Member, IEEE) received the B.S. and M.S. degrees in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea, in 2017 and 2019, respectively, where he is currently pursuing the Ph.D. degree.

His current research interests include energy-efficient multi-core architectures/accelerator application-specific integrated circuits (ASICs)/ systems, especially focused on artificial intelligence including deep reinforcement learning and computer vision, energy-efficient processing-in-memory accelerator, and deep-learning algorithm for efficient processing.

Hoi-Jun Yoo (Fellow, IEEE) graduated from the Department of Electronic Engineering, Seoul National University, Seoul, South Korea, in 1983. He received the M.S. and Ph.D. degrees in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea, in 1985 and 1988, respectively.

Dr. Yoo served as a member for the Executive Committee of ISSCC, Symposium on VLSI, and A-SSCC, the TPC Chair for the A-SSCC 2008 and ISWC 2010, the IEEE Distinguished Lecturer from 2010 to 2011, the Far East Chair for the ISSCC from 2011 to 2012, the Technology Direction Sub-Committee Chair for the ISSCC in 2013, the TPC Vice Chair for the ISSCC in 2014, and the TPC Chair for the ISSCC in 2015. http://ssl.kaist.ac.kr