International Journal of Parallel Programming (2019) 47:360–372
https://doi.org/10.1007/s10766-018-00624-9

BSHIFT: A Low Cost Deep Neural Networks Accelerator

Yong Yu[1,2,3] · Tian Zhi[1] · Xuda Zhou[4] · Shaoli Liu[1,3] · Yunji Chen[1,2] · Shuyao Cheng[5]

Received: 28 September 2018 / Accepted: 18 December 2018 / Published online: 1 January 2019

Abstract
Deep neural networks (DNNs) have become ubiquitous in artificial intelligence applications, including image processing, speech processing and natural language processing. However, the main characteristic of DNNs is that they are computationally and memory intensive, making them difficult to deploy on embedded systems with limited hardware resources and power budgets. To address this limitation, we introduce a new quantization method with mixed data structure and bit-shifting broadcast accelerator structure BSHIFT. These works together reduce the storage requirement of neural networks models from 32 to 5 bits without affecting their accuracy. We implement BSHIFT at TSMC 16 nm technology node, and the efficiency achieves 64 TOPS/s per watt in our experiments.

Keywords Deep neural networks · Low power · Lossless · Accelerator

✉ Yong Yu
yuyong@ict.ac.cn

Tian Zhi
zhitian@ict.ac.cn

Xuda Zhou
anycall@mail.ustc.edu.cn

Shaoli Liu
liushaoli@cambricon.com

Yunji Chen
cyj@ict.ac.cn

Shuyao Cheng
cheng-sy15@mails.tsinghua.edu.cn

1 The Institute of Computing Technology, The Chinese Academy of Sciences, Beijing 100190, China

2 University of Chinese Academy of Sciences, Beijing 100049, China

3 Cambricon Tech. Ltd, Beijing 100191, China

4 University of Science and Technology of China, Hefei 230026, China

5 Tsinghua University, Beijing 100084, China

1 Introduction

Neural Networks have become the dominating algorithms in a broad range of applications, including image recognition [19], object detection [21], speech recognition [1] and natural language processing [6], as they achieve state-of-the-art accuracy on many artificial intelligence tasks. However, neural networks are both computation intensive and memory intensive, which make them difficult to be deployed on the frequently used computing system such as PCs and smart phones.

Recently, researchers have proposed a number of techniques to rise to such challenges, including pruning methods [11,14,23,25] and quantization [7,12,17,20]. Pruning methods achieve remarkable reduction in storage by optimizing both the size and the generalization capabilities of neural networks, but their induced irregular sparsity is hard to implement for an accelerator. Since the available computing power and memory footprint are restricted in actual situation, precision quantization of numerical representations is commonly used for higher computing efficiency. The main problem of quantization is accuracy degradation due to its lower numerical representation.

In this paper, we propose a new quantization method mixed numerical representation to reduce the data amount while maintaining the accuracy. We also design a hardware accelerator, which is called BSHIFT. By introducing broadcast mechanism, BSHIFT can reuse input neurons as much as possible. BSHIFT supports mixed data structure which are power of 2 synapses and FlexPoint neurons [17]. The proposed accelerator features are using broadcast shift and accumulate operation replacing multiply and add (MAC) in deep neural network running on CPU/GPU/accelerators, supporting both mixture data structure in this paper and normal quantization low bit width data structure, good configurable ability.

The rest of this paper is organized as follow. We introduced the researchs background and motivation in Sect. 2. In Sect. 3, we described the mixed quantization method we used. In Sect. 4, we proposed the architecture of the BSHIFT, which can support different layer of DNNs. The experimental methodology and our experimental results are presented in Sect. 5, conclusion in Sect. 6.

2 Background and Motivation

Recently, deep neural networks is becoming more and more popular due to its capacity of pattern recognition. However, deep neural networks usually need huge computation and storage resources. For training, the resources may not be a big problem thanks to cloud computing with high performance GPUs and CPUs. The large consumption of computation and storage during training may not affect normal usage because of. In normal life, the application of deep neural networks usually targets its inference, which is the pattern recognition process in deep learning. The huge resources need is a key problem to deploy deep neural networks, especially for embedded systems.

The huge computation and storage resources in deployment of DNNs raise some problems in real applications. First, deep learning is growing fast nowadays. There

are new larger and deeper deep neural networks architecture than ever nowadays. This make deploy deep neural networks in embedded system a big challenge. Nobody want upgrade their hardware frequently because of algorithm modify. Second, the computation intensive of DNNs may consume battery power than other algorithms, which is critical to embedded systems. The large data amount in DNNs need much hardware storage and memory in run-time. The storage and memory are both limited in embedded systems. Even more, it make update DNNs model a big challenge.

For these problems, both of compressing neural networks in algorithm aspect and design DNNs accelerator become the main solutions. The convolution layers and fully connect layers in DNNs are the most computational complexity, since their complex matrix computation. Reserchers have proposed many methods to compress the parameters in these layers. Recent researches show that it is unnecessary to use 32 bits floating-point data representation in deep neural networks. Lower bit data representation not only reduces both time and energy for the memory access but also makes computation amount less [5].

Gupta et al. [12] use 16 bit integer data structure to train deep neural networks. In compare with floating-point data structure, 16 bit integer data structure reduce memory access by $2\times$ and make multiply and accumulate computation amount less on integrated circuit. But this method leads to deep neural networks accuracy loss.

To address the accuracy loss problem, [17] proposed FlexPoint data structure, aiming at a complete replacement of 32-bit floating point format in deep neural networks training and inferencing. FlexPoint data structure is designed to support modern deep neural network without modifications. FlexPoint data format is different with floating-point data structure. Floating-point has 1 bit for sign, 23 bits for mantissa and 8 bits for exponent. FlexPoint tensors have a shared exponent which is stored as integer in 5 bits and mantissa is integer number in 16 bits. During the training, FlexPoint tensors is dynamically adjusted to minimize overflows and maximize available dynamic range.

Using 16 bits data format in deep neural networks is also too much. Deep neural networks keep advancing towards larger and deeper architecture nowadays, which make it hard to implement in normal life, especially for some limited hardware. In [7], deep neural networks are implemented with binary synapses. [20] proposed the ternary synapses deep neural networks. [27] designed an incremental quantization network with power of two neurons and synapses. These quantization methods have a significant accuracy loss, especially on big dataset models like AlexNet [18], VGG [22], GoogleNet [24]. [27] implemented incremental quantization network with power of two synapses and floating-point neurons which achieves state-of-the-art accuracy. However, floating-point neurons and power of two synapses data structure can't just make neurons shift to achieve multiply operation, which benefits hardware more.

Deep neural networks have achieved the state of art results in many artificial intelligent tasks. But the computation complexity and large parameters amount have restricted the deployment in real world. In tradition, deep neural networks algorithms is usually deployed in cloud computing such as GPU servers and CPU servers. However, along with the development of embedded systems, there are a increasing number of artificial intelligent demand in embedded systems including autonomous vehicles,

smart phones, smart lights. CPUs or GPUs are no longer suitable for these embedded systems because of the power and weight limitation of embedded systems. For this reason, ASIC based hardware accelerators are seen as efficient solutions [5].

There are many new hardware architectures in deep learning accelerators. [2,3,26] designed a sequential vector dataflow structure. [4,8] designed a spatial based dataflow structure. [9] proposed a SIMD based dataflow structure. But there is less work about algorithm and hardware co-design, which means designing DNNs optimizing algorithm that benefit DNNs accelerator architecture and designing DNNs accelerator that give full play to the advantage of DNNs optimizing algorithm.

Without DNNs accuracy loss, reducing the data amount in deep neural networks makes it easy to implement on embedded system. For the purpose, we designed DNNs optimizing algorithm mixed quantization method and DNNs accelerator BSHIFT.

3 Mixed Quantization Method

In this section, we describe the mixed quantization method in detail. Our proposed method is divided into quantization on synapses and quantization on neurons according to different precision requirements. The quantization data structure of synapses is powers of 2 and the value of exponent is stored in S bits. For example, S can be 5 bits, which is one bit for zero and four bits represent at most 16 different values for the powers of two. And the quantization data structure of neurons is 16 bits integer storing mantissa and 5 bits integer storing shared exponent.

Using mixed quantization data structure, we decease data amount from 32 bits floating-point neurons and synapses to 16 bits integer neurons and 5 bits integer synapses. Instead of floating-point multiply operation, mixed quantization method only needs bit shift operation on integer number in neural networks inference. Because implementing floating-point multiply on integrated circuit is much more complex than implementing bit shift on integrated circuit. Mixed quantization method leads less power cost and smaller area in DNNs accelerator.

We implement mixed quantization method by retraining original floating-point DNNs. There are two steps in the proposed method.

Step 1: Use the proposed quantization method to train power of 2 synapses. We can start from the floating-point data structure and do the quantization incrementally. The quantization computes Gaussian distance between floating-point value in original neuron network synapses and power of 2 values. And choose the most near power of 2 value as power of 2 synapse. The proposed method is amenable for specialized training hardware optimized for field deployment of already existing deep learning models.

Step 2: Change the data structure of neurons from floating-point to shared exponent integer. After step 1, we have made a power of 2 synapses and floating-point neurons DNN. Based on maximum neuron value in the layer, we determine the value of this layer neurons' shared exponent which should cover the maximum neuron in the layer. According to shared exponent, we convert floating-point neurons to integer neurons. The convert is a shift operation on floating-point mantissa.

Table 1 Quantization performance on different DNN models

Networkd	Bit-width (synapses/ neurons)	Top-1 error (%)	Top-5 error (%)
AlexNet ref	32/32	42.76	19.77
AlexNet	5/16	42.86	19.67
VGG-16 ref	32/32	31.46	11.35
VGG-16	5/16	31.78	11.49
GoogleNet ref	32/32	31.11	10.97
GoogleNet	5/16	31.41	10.99

We use caffe [15] as our experimental framework and modify caffe to meet mixed quantization method. The synapses are represented powers of two storing exponent value in 5 bits. Neurons are stored as integer in 16 bits. Table 1 shows the performance of our quantization method on different DNN models (AlexNet, VGG-16, GoogleNet). According to our experimental results, mixed quantization method decrease data amount while achieving accuracy lossless. Mixed quantization method also replace multiply operation in DNNs to bit shift operation, which benefits DNNs accelerator much more than other binary DNN mothods.

4 Architecture

4.1 Overview

We present the proposed accelerator architecture BSHIFT as shown in Fig. 1. BSHIFT has no multiply module. The key components of our accelerator are the broadcast shift module (BSM) and continuous accumulation module (CAM), which are designed for efficient computation of core operations of neural networks, i.e., bit shift and accumulation. The memory stores the parameters and topology of neural networks. The DMA delivers the input neurons and synapses to the BSM and receives output neurons from the CAM. The control unit (CU) is the root of the accelerator and controls the whole execution process.

As mentioned in section three, the proposed data structure is designed to meet the requirements of both operational complexity and computational accuracy for AI applications.

By applying exponential representation synapses and integer neurons with shared exponents, the normal MAC computation can be converted to bit-shifting operations and accumulations in our design. The changes above can not only achieve lower energy in DNN applications but also reuse input neurons as much as possible. Mixed quantization method can faithfully maintains algorithmic parity with full-precision floating point training and supports a wide range of deep network topology.

In the following implementations, the neurons are represented by mixed quantization method, where 8 bits store data value and a 4 bits hyper parameter stores shared exponent value of all neurons in this layer. And synapses exponent are stored in 5 bits representing the power of 2.

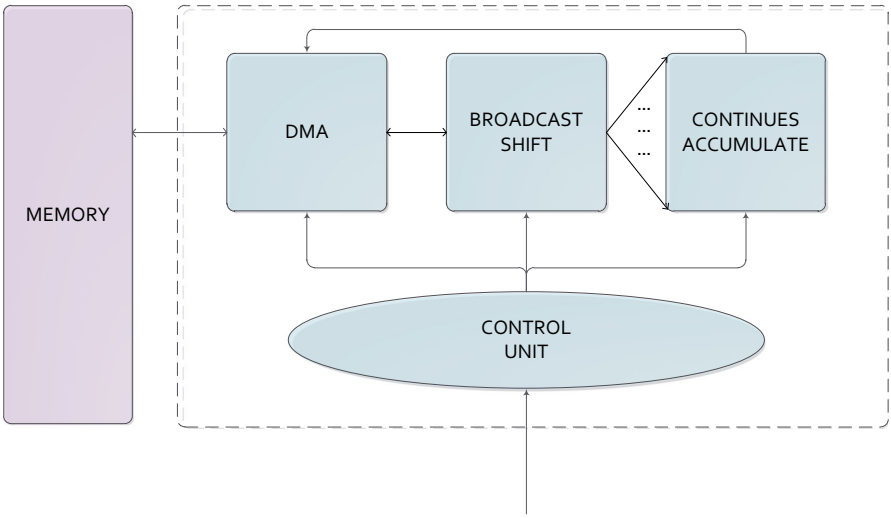

Fig. 1 An overview of BSHIFT

4.2 Computation

In deep neural networks, multiply and accumulate (MAC) operations occupy a large proportion of computation amount. The following formula (1) shows the computational relationship between neurons and synapses.

$$Neuron\_out[j] = \sum (Neuron\_in[i] * Synapses[i][j]) \tag{1}$$

As shown in the figure above, BSHIFT consists of four parts: broadcast shift modules (BSM), continues accumulate modules (CAM), direct memory access modules (DMA), control unit modules (CU). DMA is memory access module. It can receive instructions from CUs and fetch data from memory. The memory can be DRAM or SRAM, either on-chip or off-chip. Memory choose is configurable in BSHIFT. DMA also links to BSs and CAs. DMA is made to feed synapse and neuron to BSs and receive write back data from CAs. CUs can both receive control instructions and deliver them to the other modules.

As mentioned before, Broadcast Shift modules (BSs) and Continues Accumulate modules (CAs) are main components in our proposed design. By introducing broadcast mechanism, we propose a scatter-based architecture.

The inner structure of BSs and CAs is shown in the Fig. 2. The computations can be done in parallel. Broadcast Shift modules (BSs) receives input neurons one by one using pipeline technology. As shown in the formula (1), we need to multiply input neuron Neuron_in[i] by Synapses[i][j] (power of 2, only store exponent). In our design, Neuron_in[i] is fed into n channels in parallel. We use bit-shifting to replace multiplications to make sure the operations above can be done in one cycle. After bit shifting and accumulation, the results will be stored in the intermediary neurons. After Continues Accumulate module, we have N output neurons in parallel. To compute all

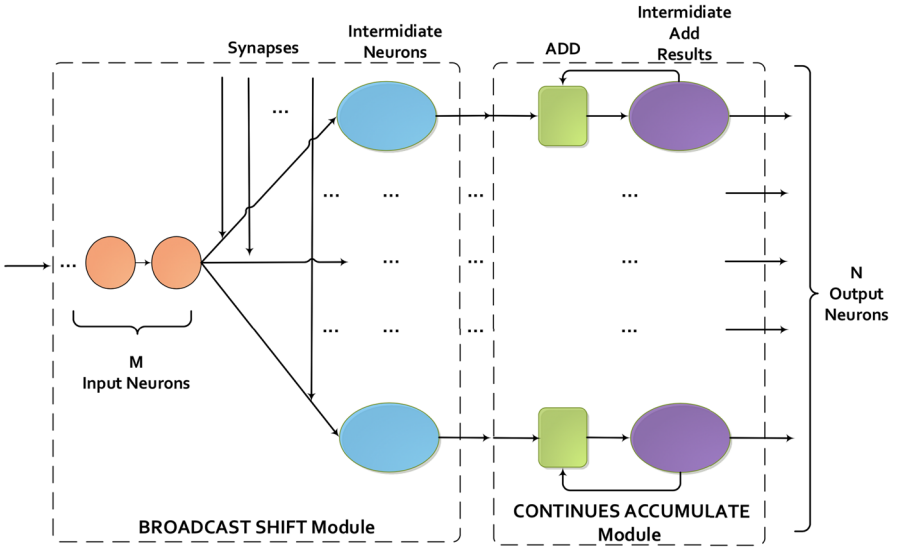

Fig. 2 Data flow in broadcast shift module and continues accumulate module

of M input neurons, CA need M cycles to accumulate the result and output final N out neurons in parallel.

The major challenge for CA module is how accumulation operation is done pipelining. It may lead pipeline bumble using one input channel calculating binary operator add. A method to eliminate the bubble in continues accumulate pipeline is shown in the following Fig. 3. By calculating accumulation intermediate results ping-pong, we solve the problem of pipeline bumble. The cycle number that add circuit needs is determined on technology node. We show two examples here to show the method we used.

For one cycle add circuit, CA stores first add number at first cycle. Next cycle, twice add number is added on first number. The result is stored. On the third cycle, third number in one channel is added on sum of first number and twice number. The following cycles are similar. CA output the final accumulation result in the end.

For two cycles add circuit, CA stores the first two add number in two registers at first two cycles. On third cycle, third number is added on the first number. Then result is stored as odd result in register. On fourth cycle, fourth number is added on twice number. Then result is stored as even result. The following cycles are similar. When the last number has come, there are odd result and even result add operation to compute final result.

4.3 Mapping

The cardinal principle for our data flow design is to reuse input neurons as much as possible. In mixed quantization data we proposed, the data width of neurons is wider than the data width of synapses. And different output neurons use the same

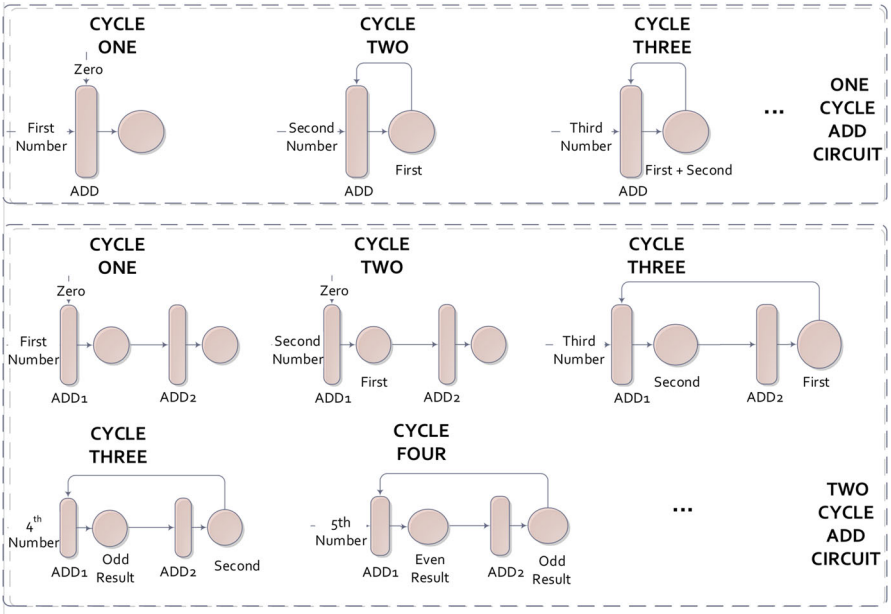

Fig. 3 Continues accumulate method

input neuron cycle by cycle. In deep neural networks, the major part of computation cost is in fully connect layer and convolution layer. The way of fully connect layer and convolution layer computation on BSHIFT is shown in Fig. 4. In fully connect layer, BSHIFT need broadcast IN times to calculate N output neurons in parallel. In Convolution layer, BSHIFT need broadcast CIN times to calculate N output neurons in parallel.

In fully connect layer, BS modules broadcast each input neuron to N output neurons. Meanwhile, shifting on the same input integer neuron with different power of 2 synapses. This operation is equal to one input neuron multiply its synapses related. Because we replace multiply operation to shift operation, it make broadcast one input neuron to multiple output on integrated circuit physical implementation possible. In multiplier circuit, there is many logic gates and wires, which make it hard to broadcast one multiplier to another. CA modules accumulate intermediate results to final output neurons. One input neuron is reused by N output neurons. In this way, we reuse input neuron as much as possible. There is no additional synapses data move.

In convolution layer, lets suppose that there are H height multiply W width input neurons in C channels. BSHIFT broadcasts C channel input neurons one by one instead of H*W channels. In compare with fully connect layer, convolution layer has less different computation patter. If we broadcast in the 2D convolution dimension, it is difficult to control CA modules. We broadcast C channel, which is also a multiply and accumulate (MAC) operation. BSHIFT is suitable to run MAC operation. For convolution layers' shared weights, they can be stored in Broadcast Shift module and be reused in the same layer. module to reduce memory access.

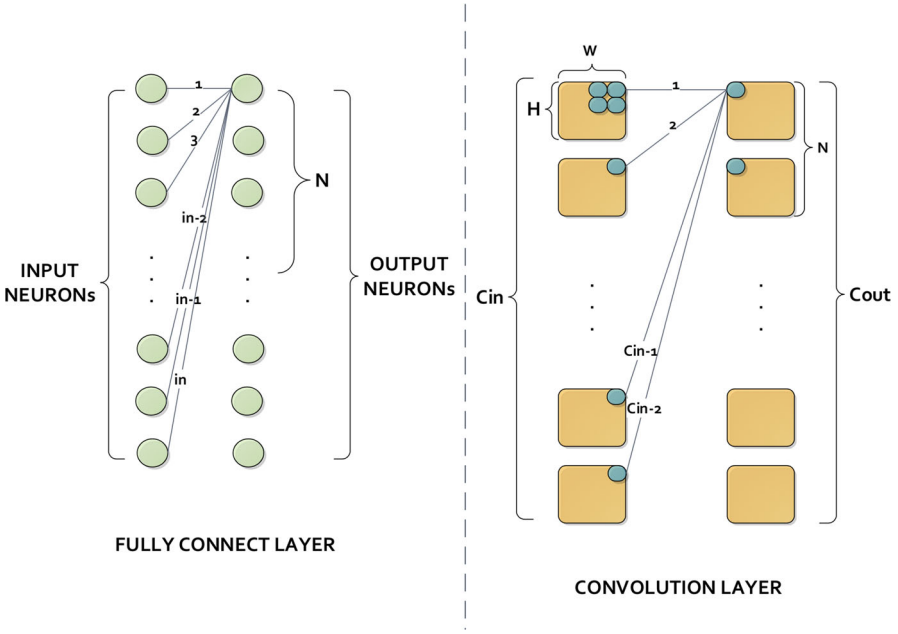

Fig. 4 Implementation of fully connect layer and convolution layer in BSHIFT

4.4 Storage

The memory stores the control instructions, network topology and parameters, i.e., neurons and synapses. We do not specify the choice of memory type, which can be DRAM, cache, scratch pad, i.e., either off-chip or on-chip. We believe the memory choice for ASICs depends on many aspects including price, application situation (throughput and latency), area, physical implement ability and so on. Without loss of generality, the memory can split into four sub-memories, which stores the input neurons, output neurons, synapses and instructions respectively.

4.5 Control

The CU decodes instructions from memory into detailed control signals for execution coordination, memory accesses and data organization. The CU monitors the states of all modules and controls them by setting the corresponding control registers. Here, we define a VLIW likely instructions set for our accelerator for high efficiency.

4.6 Configurability

We prefer to considering BSHIFT as a computation core with high configurability. Figure 5 shows some configures for different focus about computation or IO. With different topology of BSHIFT and memory, we achieve computation performance and

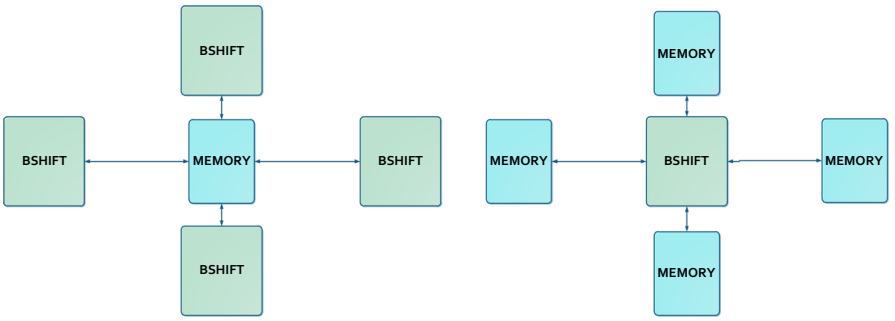

Fig. 5 Compute intensive BSHIFT and IO intensive BSHIFT

input/output throughput configurability. Some deep neural networks are computation intensive while others are memory access intensive. So users can use BSHIFT as a computation core to build their own system on chip (SOC) considering different applications.

5 Experiment

5.1 Experiment Methodology

We use CAD tools as our performance and energy measurement tools. The design has been implemented by TSMC 16 nm technology. We wrote Verilog RTL of BSHIFT where broadcast shift module has 256 output ports in our experiments. We simulated BSHIFT rtl on Synopsys VCS simulation tool. We then synthesized the design with Synopsys Design Compiler at 1 GHz clock domain. As for hardware performance benchmark, we use AlexNet in the experiment, because there are many deep learning accelerators experiment results on AlexNet.

5.2 Experiment Results

In the experiment, we use TSMC 16 nm technology under typical corner. The throughput and energy consumption comparison with state-of-the-art accelerators is shown in Table 2 [8,10,13].

Table 2 Comparison between BSHIFT with state-of-the-art accelerators

Accelerators	Power (mW)	Efficiency (GOPs/s/W)	Frequency (GHz)	Technology
NeuFlow	600	490	400	IBM45
Eyeriss	278	246	250	TSMC65
ShiDianNao	320	400	1000	TSMC65
EIE	590	5000	800	TSMC45
BSHIFT	0.82	64,000	1000	TSMC16

Table 3 Comparison equivalent efficiency in same technology

Accelerators	Equivalent efficiency (GOPs/s/W)	Frequency (GHz)	Technology
NeuFlow	2940	400	TSMC16
Eyeriss	2214	250	TSMC16
ShiDianNao	3600	1000	TSMC16
EIE	30,000	800	TSMC16
BSHIFT	64,000	1000	TSMC16

BSHIFT is a scalability architecture. The performance can be configured too. With the N is equal to 256 and 1 GHz clock cycle, our implementation of BSHIFT achieve a 256 MAC per cycle, which is equal to 512 GOPs/s. Thanks to the configurable design of BSHIFT, BSHIFT design can achieve 2*N MAC per cycle in theory.

Its not fair to compare processor architecture in different technology node. 16 nm technology node has around 6× performance per watt in compare with 45 nm technology node in deep learning applications according to [16], 9× performance per watt than 65 nm. As shown in Table 3, we calculate an equivalent efficiency in 16 nm technology. Considering the technology node, the broadcast likely deep learning accelerator architecture is also at least 2× better than other accelerator architectures in efficiency.

6 Conclusion

Based on mixed quantization method we propose, we design the broadcast likely BSHIFT to replace multiply operation. Meanwhile, using broadcast input neurons makes input neuron fully reuse and just move synapses shift locations. Furthermore, the computational complexity is much low than full precision data structure. And BSHIFT is much more ubiquitous for other AI applications than complete binary network accelerator.

There are four main contributions in this paper. First, we propose mixed quantization method which is not only ubiquitous for AI applications but also appropriate for accelerators. Second, we propose a broadcast likely architecture BSHIFT to reuse input neurons fully. Third, BSHIFT not only supports power of 2 data but also a mixture of integer neurons with shared exponent and power of 2 synapses in deep neural networks. Fourth BSHIFT can be configurable for different application situations.

Acknowledgements This work is partially supported by the National Key Research and Development Program of China (under Grant 2017YFA0700902, 2017YFB1003101), the NSF of China (under Grants 61472396, 61432016, 61473275, 61522211, 61532016, 61521092, 61502446, 61672491, 61602441, 61602446, 61732002, and 61702478), the 973 Program of China (under Grant 2015CB358800) and National Science and Technology Major Project (2018ZX01031102).

References

1. Amodei, D., Anubhai, R., Battenberg, E., Case, C., Casper, J., Catanzaro, B., Chen, J., Chrzanowski, M., Coates, A., Diamos, G.: Deep speech 2: end-to-end speech recognition in English and Mandarin. In: International Conference on Machine Learning, pp. 173–182 (2016)

2. Chen, T., Du, Z., Sun, N., Wang, J., Wu, C., Chen, Y., Temam, O.: Diannao: a small-footprint high-throughput accelerator for ubiquitous machine-learning. ACM Sigplan Not. **49**(4), 269–284 (2014)
3. Chen, Y., Luo, T., Liu, S., Zhang, S., He, L., Wang, J., Li, L., Chen, T., Xu, Z., Sun, N., et al.: Dadiannao: a machine-learning supercomputer. In: Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture, pp. 609–622. IEEE Computer Society (2014)
4. Chen, Y.H., Emer, J., Sze, V.: Eyeriss: a spatial architecture for energy-efficient dataflow for convolutional neural networks. In: ACM SIGARCH Computer Architecture News, vol. 44, pp. 367–379. IEEE Press (2016)
5. Cheng, J., Wang, P., Li, G., Hu, Q., Lu, H.: A Survey on Acceleration of Deep Convolutional Neural Networks. arXiv preprint arXiv:1802.00939 (2018)
6. Conneau, A., Schwenk, H., Barrault, L., Lecun, Y.: Very Deep Convolutional Networks for Natural Language Processing. arXiv preprint arXiv:1606.01781v1 (2016)
7. Courbariaux, M., Hubara, I., Soudry, D., El-Yaniv, R., Bengio, Y.: Binarized Neural Networks: Training Neural Networks with Weights and Activations Constrained to $+1$ or -1. arXiv preprint arXiv:1602.02830 (2016)
8. Du, Z., Fasthuber, R., Chen, T., Ienne, P., Li, L., Luo, T., Feng, X., Chen, Y., Temam, O.: Shidiannao: shifting vision processing closer to the sensor. In: ACM SIGARCH Computer Architecture News, vol. 43, pp. 92–104. ACM (2015)
9. Esmaeilzadeh, H., Sampson, A., Ceze, L., Burger, D.: Architecture support for disciplined approximate programming. In: ACM SIGPLAN Notices, vol. 47, pp. 301–312. ACM (2012)
10. Farabet, C., Martini, B., Corda, B., Akselrod, P., Culurciello, E., LeCun, Y.: Neuflow: a runtime reconfigurable dataflow processor for vision. In: 2011 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), pp. 109–116. IEEE (2011)
11. Guo, Y., Yao, A., Chen, Y.: Dynamic network surgery for efficient DNNS. In: Advances In Neural Information Processing Systems, pp. 1379–1387 (2016)
12. Gupta, S., Agrawal, A., Gopalakrishnan, K., Narayanan, P.: Deep learning with limited numerical precision. In: International Conference on Machine Learning, pp. 1737–1746 (2015)
13. Han, S., Liu, X., Mao, H., Pu, J., Pedram, A., Horowitz, M.A., Dally, W.J.: Eie: efficient inference engine on compressed deep neural network. In: 2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA), pp. 243–254. IEEE (2016)
14. Han, S., Mao, H., Dally, W.J.: Deep compression: compressing deep neural networks with pruning, trained quantization and huffman coding. Fiber **56**(4), 3–7 (2015)
15. Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., Darrell, T.: Caffe: Convolutional Architecture for Fast Feature Embedding. arXiv preprint arXiv:1408.5093 (2014)
16. Khazraee, M., Zhang, L., Vega, L., Taylor, M.B.: Moonwalk: Nre optimization in asic clouds. ACM SIGOPS Oper. Syst. Rev. **51**(2), 511–526 (2017)
17. Köster, U., Webb, T., Wang, X., Nassar, M., Bansal, A.K., Constable, W., Elibol, O., Gray, S., Hall, S., Hornof, L., et al.: Flexpoint: an adaptive numerical format for efficient training of deep neural networks. In: Advances in Neural Information Processing Systems, pp. 1742–1752 (2017)
18. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: International Conference on Neural Information Processing Systems, pp. 1097–1105 (2012)
19. Mnih, V., Hinton, G.: Learning to label aerial images from noisy data. In: International Conference on Machine Learning (2013)
20. Rastegari, M., Ordonez, V., Redmon, J., Farhadi, A.: Xnor-net: imagenet classification using binary convolutional neural networks. In: European Conference on Computer Vision, pp. 525–542. Springer, Berlin (2016)
21. Ren, S., He, K., Girshick, R., Sun, J.: Faster r-CNN: towards real-time object detection with region proposal networks. In: International Conference on Neural Information Processing Systems, pp. 91–99 (2015)
22. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. Comput. Sci. arXiv preprint arXiv:1409.1556 (2014)
23. Song, H., Pool, J., Tran, J., Dally, W.: Learning both weights and connections for efficient neural network. In: International Conference on Neural Information Processing Systems, pp. 1135–1143 (2015)

24. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1–9 (2014)
25. Wang, Y., Xu, C., You, S., Tao, D., Xu, C.: Cnnpack: packing convolutional neural networks in the frequency domain. In: Advances in Neural Information Processing Systems, pp. 253–261 (2016)
26. Zhang, S., Du, Z., Zhang, L., Lan, H., Liu, S., Li, L., Guo, Q., Chen, T., Chen, Y.: Cambricon-x: an accelerator for sparse neural networks. In: The 49th Annual IEEE/ACM International Symposium on Microarchitecture, p. 20. IEEE Press (2016)
27. Zhou, A., Yao, A., Guo, Y., Xu, L., Chen, Y.: Incremental Network Quantization: Towards Lossless CNNS with Low-Precision Weights. arXiv preprint arXiv:1702.03044 (2017)