Exploiting Near-Memory Processing Architectures for Bayesian Neural Networks Acceleration

Yinglin Zhao*§, Jianlei Yang†§, Xiaotao Jia*§, Xueyan Wang*§, Zhaohao Wang*§
Wang Kang*§, Youguang Zhang‡§ and Weisheng Zhao*§
* School of Microelectronics, Beihang University, Beijing, 100191, China.
† School of Computer Science and Engineering, Beihang University, Beijing, 100191, China.
‡ School of Electronic and Information Engineering, Beihang University, Beijing, 100191, China.
§ Fert Beijing Research Institute, BDBC, Beihang University, Beijing, 100191, China.
Email: {jianlei, weisheng.zhao}@buaa.edu.cn

***Abstract*—Bayesian inference is an effective approach to capture the model uncertainty as well as tackle the over-fitting problem in deep neural networks. Recently Bayesian neural networks (BNNs) are becoming more and more popular and have succeeded in many recognition tasks. However, the BNNs inference procedure requires numerous memory access operations due to the resulted sampling networks. In this paper, a near memory architecture is proposed for accelerating BNN inference by introducing additional memory units near the processing units. The near memory architecture could cache the frequently accessed data to reduce the data movement efficiently. Minimizing the expensive data movements between memory units and computation units contributes to cutting down the latency and energy consumption. Comparing with the traditional approach, the simulation results show that the proposed architecture reduces the energy consumption by 9% and achieves a $1.6\times$ speedup at the cost of 4% area overhead.**

***Index Terms*—Bayesian Neural Network, Near Memory Processing, Architecture.**

I. Introduction

The training of Deep Neural Network (DNN) synaptic weights usually requires large scale of training data and is susceptible to over-fitting problem [1]. The over-fitting problem may cause failures in fitting additional data or predicting future observations, which is considered as a weak suitability of a model. In addition, recent research has found that subtle pixel modifications can mis-classify the image into other labels for an image that has been correctly classified under DNN [2].

In order to overcome these shortcomings, a novel method called Bayesian Neural Network (BNN) is proposed [3]. BNN not only provides a consistent framework for statistical pattern recognition, but also has many practical advantages e.g., avoiding over-fitting problems. By exploiting the uncertainty and making use of prior knowledge, the computing pattern of BNN is closer to real circumstance, which makes it exhibit required robustness for future decision. For example, the authors of [4] propose a novel method of propagating the uncertainty through the sparsity-promoting layers and design a Bayesian Learned Iterative Shrinkage-Threshold network. Reference [5] introduces a python probabilistic programming library for Bayesian deep learning to conjoin the complimentary advantages of Bayesian methods and deep learning.

However, the BNN method faces a critical technical challenge due to the high cost on computation and data movement. While performing the inference procedure, the sampled weights are read from off-chip memory for computation and then the results are written back. Lots of data movement introduces a lot of energy consumption to perform feed-forward propagation [6]. As remarkable performance is always achieved at the cost of a high computation complexity associated to the deep layer structure, high performance computing resources and large data transmission bandwidth are necessary to accelerate the BNN inference. The works [7], [8] aim to reduce data transmission and accelerate computation procedure from a different view. With requirements of inference accuracy increasing, the energy consumption cost is becoming even higher.

Therefore, a lot of state-of-the-art researches are focused on coming up with innovative computing architectures to overcome these limitations. The traditional approach is moving data from memory all the way up to cache and then processing them in computation units. In contrast, near-memory computing (NMC) aims to perform computation close to where the data resides. The data-centric approach proposed in [9] couples computation units close to memory and seeks to minimize the expensive cost of data movement.

In view of the above, we propose a customized computation architecture to improve the computation efficiency of BNN algorithm. By integrating a small capacity memory with computation units, the frequently accessed data can be cached in the local memory that is near to the computation units, so that it could be processed more efficiently. The proposed scheme enables data reuse and reduces data transmission. The main contributions of our work are summarized as follows:

- A novel near-memory computing architecture is proposed to perform fast and energy efficient BNN inference. By taking the advantage of the low access latency characteristic of near-memory computing, the BNN inference procedure could be effectively accelerated.
- The proposed architecture has been evaluated by simulations. The results indicate that the proposed architecture could reduce energy consumption by 9% and achieve a $1.6\times$ speedup at the cost of 4% area overhead.

978-1-7281-3391-1/19/$31.00 ©2019 IEEE
DOI 10.1109/ISVLSI.2019.00045

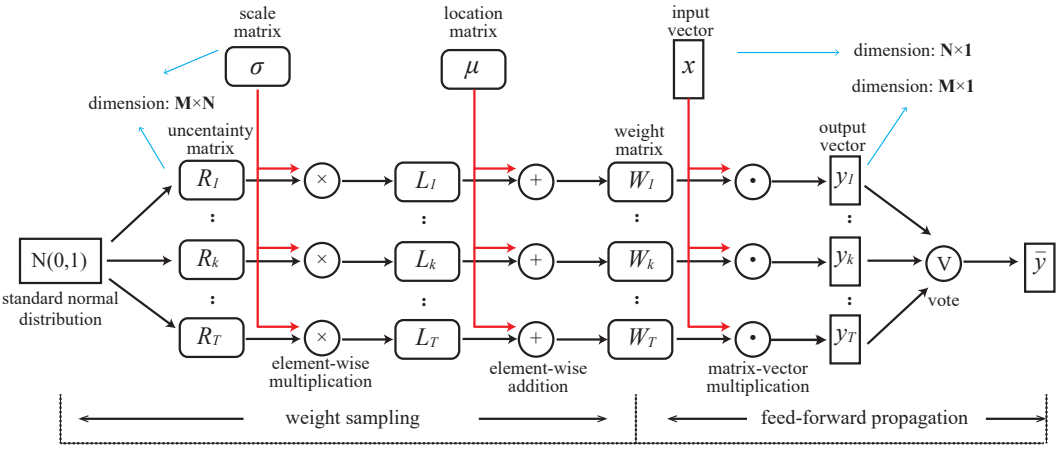

Fig. 1: Single-layer BNN dataflow is divided in two steps: weight sampling and feed-forward propagation.

The rest of this paper is organized as follows. Section II discusses some preliminaries and related work. Section III demonstrates the proposed near-memory computing architecture in detail. Section IV provides the evaluation methodology and results. And Section V concludes this paper.

II. Preliminaries

BNNs are comprised of a probabilistic model and a neural network [10]. Neural networks have universal continuous function approximation ability, and statistical models allow us to generate cases according to the posterior distribution parameters that are derived from observations. In the prediction phase, the statistical model generates a complete distribution and guarantees the probability of prediction. BNN is therefore the combination of neural networks and stochastic models, and stochastic models form the core of this integration. By introducing the augmentation of standard neural networks with posterior inference, BNN creates a deep learning framework which considers the effect of parameter uncertainty.

BNN training is related to the process of finding the posterior distribution over model parameters, which is very difficult since the posterior weight distribution is highly complex. In order to improve the efficiency of BNN training, some related BNN researches are conducted. The work in [11] introduces an easy-to-implement stochastic variational method that can be applied to most neural networks and revisits several common regularization from a variational perspective. In addition, a Turing-complete probabilistic programming language, named Edward framework [12], is adopted to train the BNN.

After the parameters are fully trained, the BNN is used for inference which could also be called prediction. The BNN prediction process requires Gaussian random variable sampling for all posterior distributions identified in the training process. Once all the random weights are obtained, BNN could start the process using the sampled parameters and inputs.

The essence of BNN algorithm is to perform repetitive forward-propagation with different sampled parameters to determine the final outputs. However, due to the data complexity and diversity, numerous hardware resources are required during data transmission and calculation. It is difficult to improve the BNN inference efficiency for traditional computing-centric architecture due to the limited bandwidth between computation units and memory units. This paper analyzes the dataflow in BNN prediction and proposes a specific computing architecture which integrates memory units near computation units to improve the data transformation efficiency.

III. Near-Memory Computing Architecture

In this section, we first describe the dataflow of the standard BNN. Subsequently, the proposed near-memory computing architecture is introduced in detail. For simplification, The single-layer BNN is used as an example.

A. BNN Inference Dataflow Analysis

The BNN inference adopts a well-trained model for prediction. With the trained parameters, the weights sampling procedure could directly affect the inference efficiency. Different from DNNs, whose weight values are deterministic, the weight value of BNN needs to make use of the uncertainty characteristics of the posterior distribution. The final result is determined by considering all outputs produced according to each sample, in which way the uncertainty is characterized. Under certain conditions, the more weights sampled, the better the uncertainty is characterized, then the final result will be more accurate.

Fig.1 shows the BNN inference dataflow of a single-layer fully connected neural network, where σ and μ are well-trained BNN weight parameter matrices. The symbol $\boldsymbol{x}$ denotes the input vector. The considered BNN contains N input neurons and M output neurons. μ and σ are posterior distribution parameters of the $M \times N$ dimensional BNN weight values. R is random value matrix whose elements are sampled from standard Gaussian distribution. The weight matrix W is calculated using R, σ and μ based on scale-location transformation. In Fig.1, $\times$, $+$ and $\cdot$ with cycle represent the operation of element-wise multiplication, element-wise addition and

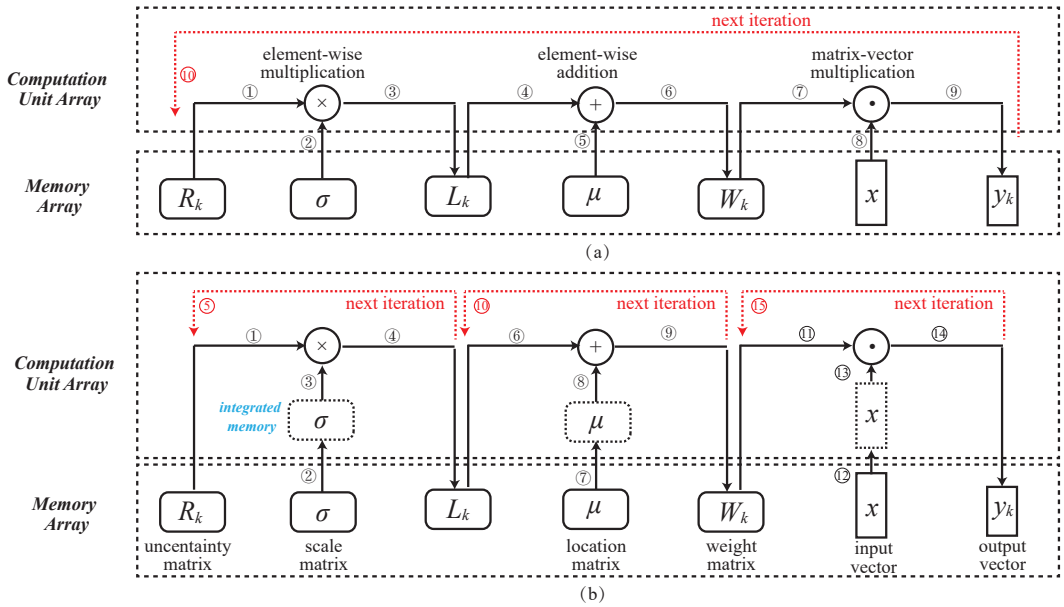

Fig. 2: The BNN inference flow. (a) Traditional computing architecture. (b) The proposed near-memory computing architecture.

matrix-vector multiplication, respectively. Based on the neural network theory, a sample output based on BNN is calculated according to Eqn.(1). T is the number of samples to be computed, and it determines the number of times that Eqn.(1) is executed in the network. In addition, the bias terms are not taken into account in Fig.1 for simplify.

$$\boldsymbol{y} = W\boldsymbol{x} + \boldsymbol{b} \tag{1}$$

B. Near-Memory Computation Architectures

Figure 2(a) shows the BNN inference dataflow in a traditional architecture. The data streaming is read from memory and send to data bus which is labeled as ①-⑨ in each iteration. When one loop is completed, the output $\boldsymbol{y}$ of one sample is obtained, which can also be considered as one iteration is finished. Then the next iteration continues. In traditional computing architecture, the operations of both data access and write back require the data bus to be activated for transferring data. Obviously, the memory access and data transferring usually account for most of the latency and energy consumption in entire system. As mentioned previously, the number of samples T determines the BNN inference accuracy. Under certain conditions, larger T brings more data for making a decision, and better inference accuracy can be achieved.

However, an important observation is that the data are usually partially accessed and reused among different iterations so that the repeated data transferring should be recognized as a waste of latency and energy. For example, the mean value μ and standard deviation σ are utilized in each iteration so that they are considered to be cached in an additional local memory close to computation units. And consequently such a caching mechanism could reduce the data transmission cost significantly. The introduced local memory is very close to computation units so that the near-memory computing is reasonably enabled. The proposed near-memory computing dataflow is illustrated in Figure 2(b). The dataflow labeled ①-⑮ represent the iterative procedures. Different from the traditional approach, the frequently accessed data μ, σ, and x are pre-cached in additional local memory when necessary until their life-cycle is end.

Considering the required silicon area efficiency, the capacity of the additional local memory is somewhat limited. Excessive memory array squeezes the area of computing units and aggravates the complexity of the control signals. It can not only increase corresponding data access latency and power consumption, but also reduce the performance improvement brought by our scheme. If the additional local memory is too small, either the R data imported from the data bus once cannot be completely processed or the data bus is not fully occupied. And it results in the waste of computation or memory resources. The number of computation units and the bandwidth of data bus should be taken into trade-off considerations for determining the size of the additional local memory.

In addition, the sizes of the matrices are different under different applications. When adopting our architecture for different applications, the size of the matrices and additional local memory may be inconsistent, which may cause that the whole matrix cannot be fully cached. But each element of μ and σ matrix corresponds to unique element in R so that they can be divided into sub-matrices and processed respectively.

IV. Evaluation

A. Experimental Setup

In our experiment, we established a three-layer fully connected neural network (784-200-200-10) based on the Mixed

TABLE I: Occupied Resources Percentage of Computation Units (CU) and Memory Units (MU).

Method	Area		Energy		Time	
	CU	MU	CU	MU	CU	MU
Trad.	55.4%	44.6%	87.9%	12.1%	14.3%	85.7%
Prop.	53.2%	46.8%	91%	9%	20%	80%

National Institute of Standards and Technology database (MNIST), which is a large database of handwritten digits commonly used for training various image processing systems. In addition, ten classes were used, and the network is trained with 20 epochs with Edward library [12]. Meanwhile, we set count T as 100 in each layer. The BNN inference is evaluated and compared on the standard implementation and the proposed near-memory architecture implementation.

And some additional principles are adopted. First, we only focus the inference process in this work and assume that the BNN training is already finished outline using Edward framework. Second, the energy consumption of generating random numbers is not taken into consideration.

B. Implementation Results

The implementations are evaluated by dividing the system into computation units part and memory units part for comparison. The core blocks are designed with Verilog language and synthesized by Synopses Design Compiler on $45\ nm$ FreePDK technology. The memory area and energy consumption are estimated by CACTI tool [13]. The evaluation is based on 8-bit fixed-point number accuracy for BNN inference. Table I describes the percentage of computation units and memory units in area, energy and latency consumption. With introducing the additional memory, the percentage of memory units area increases from 44.6% to 55.4%. As the computation units take most of energy consumption for activating adders and multipliers, the data movement reduction could slightly reduce the percentage of memory units energy cost. In addition, because the memory access operation is slower than computation, the reduction in memory access operation could bring benefits for BNN acceleration, which results in the percentage decreasing of memory units latency.

TABLE II: Comparison Results of Different Implementations.

Method	Accuracy	Area (mm^2)	Energy (μJ)	Runtime (μs)
Traditional	95.42%	5.76	172	392
Proposed	95.42%	5.98	157	245

Table II describes the performance comparisons in terms of accuracy, area, energy consumption, and the total execution time. Compared with the traditional method, our design requires an increase of 4% of the area. However, the additional memory could cache the frequently accessed data to reduce the average access consumption and latency efficiently. As shown in Table II, the proposed approach could achieve 9% reduction in energy consumption and a speedup of $1.6\times$ when evaluated on MNIST dataset.

V. Conclusion

This paper analyzes the BNN inference dataflow and clarifies the data access characteristics. And subsequently a BNN customized near-memory computing architecture is proposed to optimize the data access patterns. The proposed architecture brings a cut down in average access latency and energy consumption, which accelerates the BNN process on the premise that BNN accuracy is not affected significantly. The proposed architecture is evaluated on the MNIST dataset. The experimental results show a 9% reduction in the energy consumption and a $1.6\times$ speedup at the cost of 4% area overhead. In the future, we will improve our work from the perspective of improving the data scheduling.

Acknowledgement

This work was supported in part by the National Natural Science Foundation of China (61602022, 61501013, 61571023, 61521091 and 1157040329), State Key Laboratory of Software Development Environment (SKLSDE-2018ZX-07), National Key Technology Program of China (2017ZX01032101), CCF-Tencent IAGR20180101 and the International Collaboration Project under Grant B16001.

References

[1] C. M. Bishop, *Pattern recognition and machine learning*. springer, 2006.

[2] A. Nguyen, J. Yosinski, and J. Clune, "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images," in *Proceedings of CVPR*, 2015, pp. 427–436.

[3] Y. Gal and Z. Ghahramani, "Bayesian convolutional neural networks with bernoulli approximate variational inference," *arXiv preprint arXiv:1506.02158*, 2015.

[4] D. Kuzin, O. Isupova, and L. Mihaylova, "Bayesian neural networks for sparse coding," in *Proceedings of ICASSP*, May 2019, pp. 2992–2996.

[5] J. Shi, J. Chen, J. Zhu, S. Sun, Y. Luo, Y. Gu, and Y. Zhou, "Zhusuan: A library for bayesian deep learning," *arXiv preprint arXiv:1709.05870*, 2017.

[6] B. Reagen, P. Whatmough, R. Adolf, S. Rama, H. Lee, S. K. Lee, J. M. Hernández-Lobato, G.-Y. Wei, and D. Brooks, "Minerva: Enabling low-power, highly-accurate deep neural network accelerators," in *Proceedings of ISCA*, 2016, pp. 267–278.

[7] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Proceedings of NIPS*, 2015, pp. 1135–1143.

[8] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, "Exploiting linear structure within convolutional networks for efficient evaluation," in *Proceedings of NIPS*, 2014, pp. 1269–1277.

[9] G. Singh, L. Chelini, S. Corda, A. J. Awan, S. Stuijk, R. Jordans, H. Corporaal, and A.-J. Boonstra, "A review of near-memory computing architectures: Opportunities and challenges," in *Proceedings of DSD*, 2018, pp. 608–617.

[10] T. Auld, A. W. Moore, and S. F. Gull, "Bayesian neural networks for internet traffic classification," *IEEE Transactions on neural networks*, vol. 18, no. 1, pp. 223–239, 2007.

[11] A. Graves, "Practical variational inference for neural networks," in *Proceedings of NIPS*, 2011, pp. 2348–2356.

[12] D. Tran, M. D. Hoffman, R. A. Saurous, E. Brevdo, K. Murphy, and D. M. Blei, "Deep probabilistic programming," *arXiv preprint arXiv:1701.03757*, 2017.

[13] N. Muralimanohar, R. Balasubramonian, and N. Jouppi, "Optimizing nuca organizations and wiring alternatives for large caches with cacti 6.0," in *Proceedings of Microarchitecture*, 2007, pp. 3–14.