Parallelism in Deep Learning Accelerators

Linghao Song
Duke University
linghao.song@duke.edu

Fan Chen
Duke University
fan.chen@duke.edu

Yiran Chen
Duke University
yiran.chen@duke.edu

Hai "Helen" Li
Duke University
hai.li@duke.edu

Deep learning is the core of artificial intelligence and it achieves state-of-the-art in a wide range of applications. The intensity of computation and data in deep learning processing poses significant challenges to the conventional computing platforms. Thus, specialized accelerator architectures are proposed for the acceleration of deep learning. In this paper, we classify the design space of current deep learning accelerators into three levels, (1) processing engine, (2) memory and (3) accelerator, and present a constructive view from a perspective of parallelism in the three levels.

I. Introduction

Deep learning approaches [1], [2] have become the core of artificial intelligence (AI) research across a broad range of applications, including computer vision [3], [4], [5], healthcare [6], [7], [8], and scientific computing [9], [10], [11]. Representative Deep Neural Network (DNN) and Convolutional Neural Network (CNN) applications are normally parameter-rich and computation-intensive, which poses significant challenges to the computing power and memory bandwidth of the underlying computing platforms. However, we are approaching the end of the scaling of Moore's Law [12], and general-purpose platforms such as CPUs and GPUs will no longer benefit from the integration of cores [13]. Therefore, a novel architecture paradigm equipped with domain-specific accelerators is proposed as a common solution to provide sustainable performance and efficiency improvement for deep learning applications.

The fundamental computational component in DNNs and CNNs is Matrix-Vector Multiplication (MVM), which exhibits inherent parallel processing capability. In order to achieve high performance and energy efficiency, various accelerators are proposed to exploit the parallelism in deep leaning algorithms. Based on the parallelism being explored, we classify the design space of current deep learning accelerators into three levels as described below:

- Processing Engine Level Parallelism (PELP) comes naturally from the fact that the massive multiplication and accumulation operations in MVM can be processed independently. In this level, operation primitives are organized in a high-parallel fashion to exploit the temporal and/or spatial parallelism.

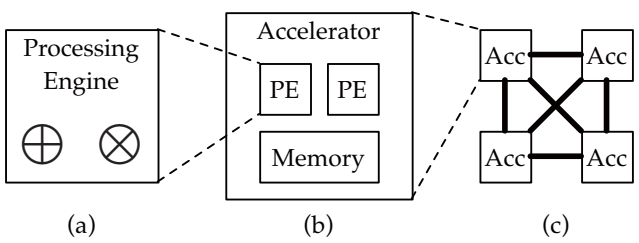


Fig. 1: Three levels of design space for deep learning accelerators. (a) processing engine, (b) memory and (c) accelerator.

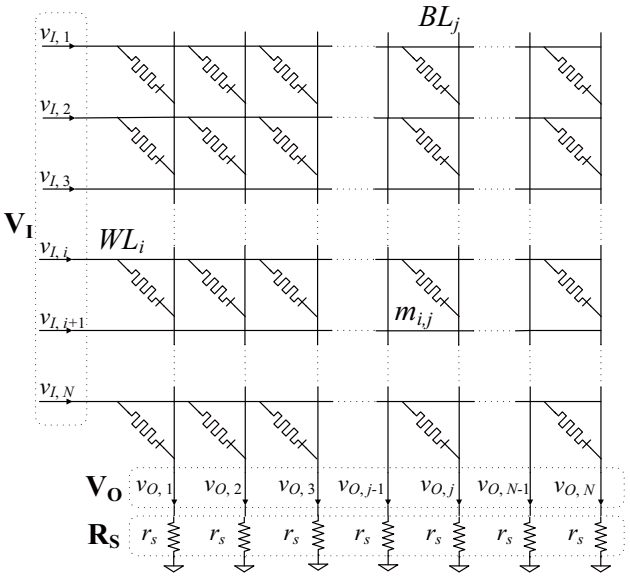


Fig. 2: Mapping a weight matrix to a ReRAM crossbar array to build a parallel VMM engine in [14].

- Memory Level Parallelism (MLP) is achieved through parallel and efficient memory accessing to alleviate the memory wall effect. In this level, the data movement and sharing between Process Engines (PEs), while memory is optimized for parallel accessing to disjoint memory space and high data utilization.
- Accelerator Level Parallelism (ALP) emerges as a recent research direction to address the the coordination of multiple accelerators in a heterogeneous system. In this level, parallelism for multiple accelerators is proposed.

In the following sections, we study the current landscape of deep learning accelerators and present a constructive view from a perspective of parallelism in the above three levels.

II. Parallelism in Processing Engines

A. Matrix-Vector Multiplication Engine

Matrix-Vector Multiplication (MVM) is the basic computation type in DNNs. The time complicity for the multiplication between an N-by-N matrix and an N-by-1 vector is $O(N^2)$ in a conventional single thread processing unit. An MVM engine processes the multiplications and accumulations in MVM to achieve a lower time complexity than $O(N^2)$.

Resistive random access memory (ReRAM) [15] is a promising candidate for MVM processing engine design because of the characteristics of high reading speed, high density and multi-level cells. Hu *et al.* [14] proposed to utilize a ReRAM crossbar array to design an MVM processing engine to calculate $\mathbf{y} = \mathbf{W} \times \mathbf{x}$. As shown in Fig. 2, the matrix weight $m_{i,j}$ is mapped to the conductance state of the cell

978-1-7281-4123-7/20/$31.00 ©2020 IEEE

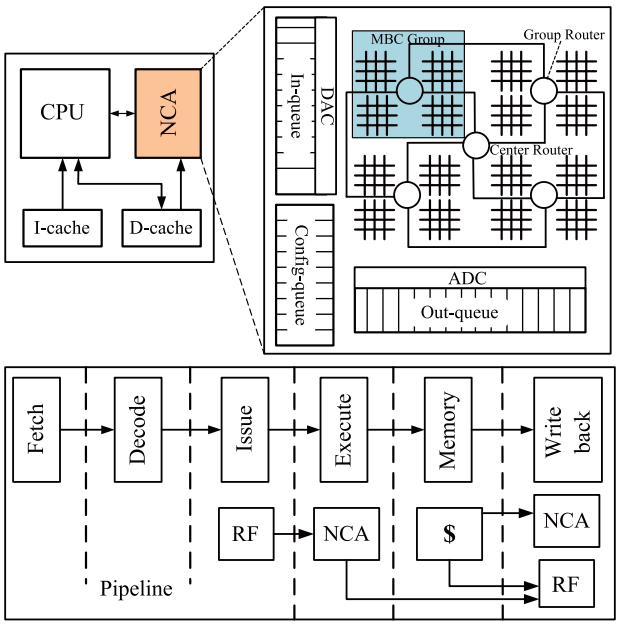


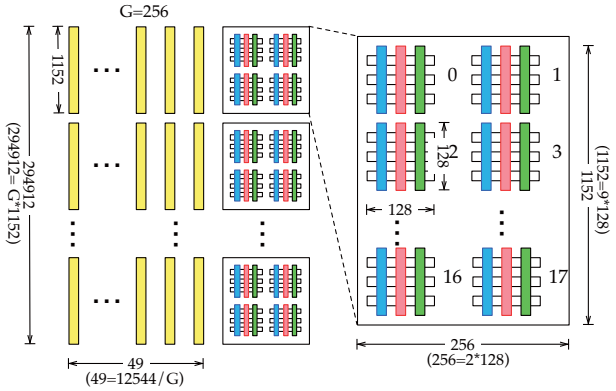
Fig. 3: An on-chip processing engine design for the acceleration of MLP and AAM [17], [18].

$g_{i,j}$ at the ith row and jth column in a ReRAM array. In computation, the input vector $\mathbf{x}$ is encoded as the voltage levels $\mathbf{v}_i$ and applied on the wordlines. For each individual cell, the multiplication is performed as the conversion of voltage to current, *i.e.*, $\mathbf{i}_{i,j} = \mathbf{v}_i \cdot g_{i,j}$. On each individual bitlines, the currents from all the connected ReRAM cells are accumulated, *i.e.*, $\mathbf{i}_j = \sum_i \mathbf{i}_{i,j}$. To get the results of the MVM, we just need to sense the currents from the bitlines to get the output vector $\mathbf{y}$. In the ReRAM MVM processing engine, on each bitline, N multiplications are computed in parallel, and for the whole array, the N bitlines are performing the accumulation in parallel. The parallel multiplication and parallel accumulation result in a $O(1)$ time complexity for a matrix-vector multiplication. Furthermore, in 2016, Hu *et al.* [16] fabricated a 4×4 ReRAM MVM processing engine.

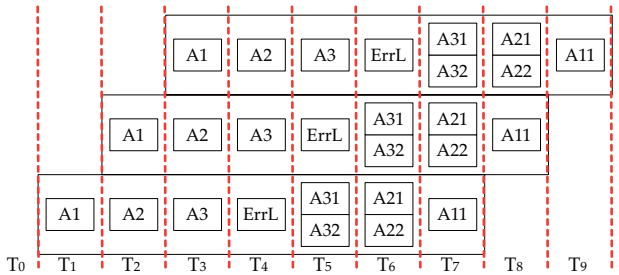
B. On-chip Processing Engine

With the ReRAM MVM processing engines, a larger scale on-chip design can be built for more complex tasks than matrix-vector multiplication. Harmonica [17], [18] is an on-chip accelerator for Multilayer Perceptron (MLP) and Auto-Associative Memory (AAM) applications. Because the scale of the architecture and the applications is relative small compared to typical DNNs, we classify Harmonica as an on-chip processing engine.

In Harmonica, groups of ReRAM MVM processing engines are deployed as shown in Fig. 3. Between the ReRAM MVM processing engines, an analog Network-on-Chip (NoC) is used to transfer intermediate data. The Digital-to-Analog-Converter (DAC) with an in-queue and the Analog-to-Digital-Converter (ADC) with an out-queue are deployed for data interface. A config-queue is used to store the routing information. The NoC coordinates the input signal to the ReRAM MVM processing engines and collect computing results from the ReRAM MVM processing engines to out-queue. Thus the Harmonica architecture benefits from the parallel execution



(a) Intra-layer parallelism.



(b) Inter-layer pipeline.

Fig. 4: (a) Intra-layer parallelism and (b) Inter-layer pipeline in PipeLayer [19].

of multiple matrix-vector multiplications. To further explore instruction level parallelism, an Instruction Set Architecture (ISA) is designed for Harmonica.

The processing of MLP requires multiple cascaded ReRAM MVM processing engines and the processing of AAM requires recurrent data delivered back to the same ReRAM MVM processing engines. The model sizes of two applications are relatively small. For large scale applications especially convolutional neural networks, on-chip processing engines are not capable. Stand-alone accelerators are required.

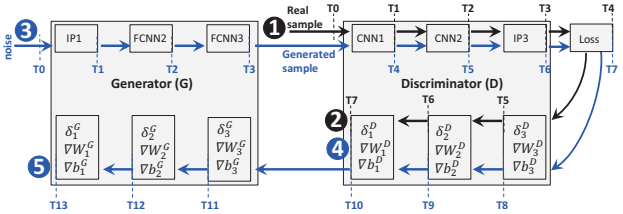
III. Memory Level Parallelism

A. Parallelism in Accelerators for DNN Training

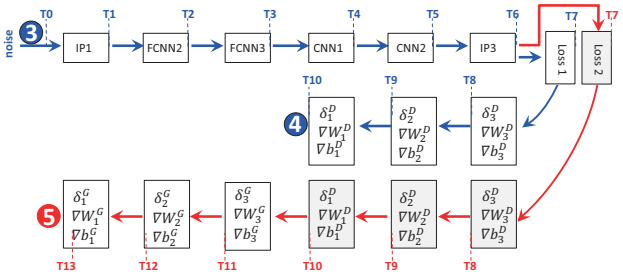
Stand-alone accelerators are designed for large-scale deep learning applications. In DNN training, massive parallel processing engines are deployed and data accessing to memory is delicately optimized. The basic idea for parallelism in accelerators for DNN training is to enable memory level parallelism for the regular sequential data layout in DNNs to achieve high throughput.

CNN Training Accelerator

In PipeLayer [19], intra-layer parallelism and inter-layer pipeline are the two schemes for parallel processing. The input feature maps of the convolutional layers are converted from a three-dimensional tensor into a Toeplitz matrix format to compatible with the MVM processing engines and fully utilize the parallel matrix-vector multiplication. For intra-layer parallelism, the weight for one layer is duplicated and multiple processing engines are mapped with the same weight as shown in Fig. 4a. Each processing engine is also equipped with a buffer memory which stores the corresponding input feature maps. As a result, processing engines are accessing



(a) Training paths in GAN.



(b) Improved training procedure by computation sharing.

Fig. 5: (a) Training paths in GAN and (b) Improved training procedure by computation sharing [20].

memory in parallel. The weight and feature map of each layer in a network is assigned to a group of processing engines and a buffer memory space. If we consecutively execute the computation for each layer, when the computation resource and memory for one layer is busy, the computation resource and memory for other layers is idle. So we propose the inter-layer pipeline which is a layer-wise pipeline design as shown in Fig. 4b to fully unitize the processing engines and memory. One layer is scheduled as one stage in the pipeline and the buffer memory coordinates data movement between two layers. Thus, multiple input data are processed in parallel to achieve a high throughput.

GAN Training Accelerators

Generative Adversarial Networks (GANs) [21], [22] have demonstrated a great opportunity toward next generation of unsupervised deep learning. In a GAN model, a generator (G) and a discriminator (D) are simultaneously trained against each other via an adversarial process. A generator captures the data distribution and attempts to generate synthetic samples, while a discriminator implements a binary classifier to differentiates the samples generated by a generator against real samples. This learning process is performed iteratively until we receive a generator with strong generative capability and a discriminator with high classification accuracy. Fig. 5a illustrated the training paths in a GAN, which involves three dependent paths: (1)❶~❷ depicts the dataflow of training D on real training samples; (2) ❸~❹ shows the path of training D on generated samples; and (3) ❸~❺ illustrates the training phase of G. Clearly, in addition to the parallelism in a single network training explored in previous works [19], [23], GAN also exhibits the parallelism between the training of multiple deep network models (ie, generators and discriminators). Therefore, a customized accelerator architecture is needed to accommodate the parallelism between the two DNN models, thereby improving the performance and energy efficiency in GAN training.

ReGAN [20] exploited the similar pipelined training procedure as in [19] to reduce on-chip memory access. In particular,, ReGAN presented *Spatial Parallelism* and *Computation Sharing* to parallel the multiple training phases to further improve performance. *Spatial Parallelism* co-process the calculations in ❶~❷ and ❸~❹ by duplicating D for multiple copies, while *Computation Sharing* parallel phases ❸~❹ and ❸~❺ since they share the same forward path as demonstrated in Fig. 5b. In addition to the multiple training phases involved in GAN, another important reason why existing solutions are unable to efficiently support GAN training is due to the fact that GAN utilizes a new operator, called transposed convolution (TCONV), which introduces significant resource underutilization as it inserts massive zeros in its input before a convolution operation. RED [24] presented *pixel-wise mapping* and the *zero-skipping* for TCONV inference acceleration. The pixel-wise mapping scheme is able to eliminate zero-inserting operations in TCONV and improve the resource utilization. The zero-shipping data flow increases the computation parallelism and further improve computing efficiency. ZARA [25] proposed a novel computation deformation technique that can skip zero-insertions in TCONV. A dataflow mapper and an operation scheduler were also implemented to support the proposed execution model. These optimized zero-aware computing model coupled with high-parallel architecture proposed in [20] provide significant system performance improvement compared with previous accelerators designed for general CNNs [19].

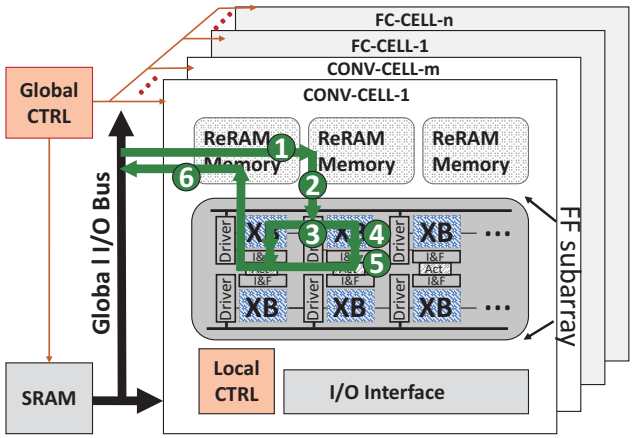


Fig. 6: Heterogeneous computing cells in EMAT [26] architecture.

Transfer Learning Accelerator

Transfer Learning [27] recently emerges as a more practical and efficient training paradigm that re-utilizes a developed neural network onto a different domain/task, significantly reducing the extensive efforts in training and data labeling in supervised learning. Chen *et al.* [26] proposed EMAT to accommodate the heterogeneous computing phases involved in transfer learning: the transferred CNN layers are fixed and execute only the feed-forward function; while the newly added FC layers are trained through back propagation. Similar with previous works, EMAT utilizes the energy-efficiency of ReRAM array for MVM and realizes a hierarchical reconfigurable design to incorporate the data patterns in transfer learning. As demonstrated in Fig. 6, two types of computation components,

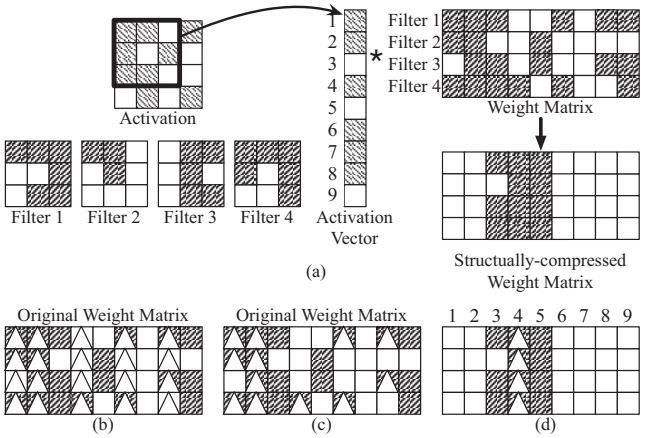


Fig. 7: (a) The elements in the feature map are converted into a vector in processing, (b) the nonzero-neuron oriented computation, (c) the nonzero-neuron and nonzero-weight oriented computation, and (d) the computation in accelerators for structural compression [29]. The performed computation is marked by triangles.

namely CNN-CELL and FC-CELL, are specialized designed according to the computation and storage characteristics of CNNs and FCs, respectively. In real-life scenario, it often desires the real-time execution of multiple tasks and dynamic adaptation capability [28]. Hence, EMAT also introduced a time-multiplexed training flow for efficiently executing multi-tasks that share a same trained CNN.

B. Parallelism in Accelerators for DNN Inference

DNN accelerators for inference are usually constrained by hardware resources and stringent power budgets. Model compression methods such as weight sparsity and connection pruning can significant reduce the scale of model size and hence, reduce the computation for DNN inference. However, conventional element-wise compression and pruning is not helpful for inference acceleration because of the irregularity of the indexing and accessing of the element-wisely compressed data. Accelerators for structural compression with regular, sequential and light memory accessing are needed.

Accelerator for Structural Compression

The acceleration of DNN inference takes the benefit from zero-element skipping. Two element wise skipping schemes are (1) nonzero-neuron oriented computation where the computation is performed only when the neuron element is nonzero, and (2) nonzero-neuron and nonzero-weight oriented computation where the computation is performed only when both the neuron element and the weight element are nonzero, as shown in Fig. 7 (b) and (c). However, because data are element-wisely compressed in the two schemes, massive irregular and random memory accessing is incurred, which significantly hinders the memory level parallelism. At the same time, element-wisely compression requires index processing for every element, which increase the computation. Thus, structural compression is required for memory efficient DNN inference, as shown in Fig. 7 (d). For the structurally compressed weight, within a weight chunk, elements are placed continuously thus the accessing is sequential. Because

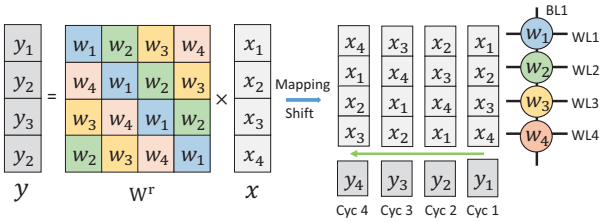


Fig. 8: Shift method and overall mapping scheme [30].

the chunks usually have the same size, thus prefetching and parallel memory accessing can be utilized. Furthermore, the structural compression chunks can be set to fit the hardware buffer, thus high memory bandwidths utilization can be achieved.

Accelerator for Block-Circulant Neural Networks

The recent proposed block-circulant DNNs [31] provide a candidate solution for efficient deployment DNNs on edge devices with controllable compression ratio and storage savings. The key idea is to approximate the original weight matrix by a circulant matrix defined by a representative vector. In this way, the original O(n^2) storage complexity is reduced to O($n\log n$). The state-of-the-art accelerators for block-circulant DNNs relied on the the Fast Fourier Transform (FFT) and Inverse Fast Fourier Transform (IFFT) for the computation. The memory level parallelism is realized by simultaneously processing massive FFT/IFFT. REBOC [30] make an important observation that FFT-base approach does not always leads to practical computation reduction. In this work, the processing of block-circulant DNNs is directly performed in ReRAM based on a novel mapping scheme with shift method as shown in Fig. 8. To fully utilize the massive parallelism for MVM in ReRAM, a block-circulant DNN model is mapped onto ReRAM crossbar with horizontal weight slicing and intro-crossbar weight duplication to achieve high crossbar utilization.

IV. ACCELERATOR LEVEL PARALLELISM

The scales of deep learning applications are becoming larger because the network model contains a larger number of parameters and the size of the data set increases to terabyte-level. As a result, computation and the data for the weight and feature map generated in the processing are no longer capable for only one accelerator. For a higher system performance, we need to consider to deploy multiple accelerators for the processing of a large-scale deep learning application. Thus, we need to propose accelerator level parallelism for the deployed multiple accelerators. In large-scale deep learning with multiple accelerators, the data communication between accelerators becomes the bottleneck. We are focusing to design an accelerator level parallelism to reduce the communication of weight and the feature map transferred between accelerators in [32].

In DNN training, there are three tensor computation for each layer, *i.e.*, forward, backward and gradient. In forward, the input feature map tensor $\mathbf{F}_l$ is multiplied with the weight tensor $\mathbf{W}_l$ and then an element-wise activation is applied to get the output feature map tensor $\mathbf{F}_{l+1} = f(\mathbf{F}_l \otimes \mathbf{W}_l)$. In backward, the error tensor $\mathbf{E}_{l+l}$ the transposed weight tensor

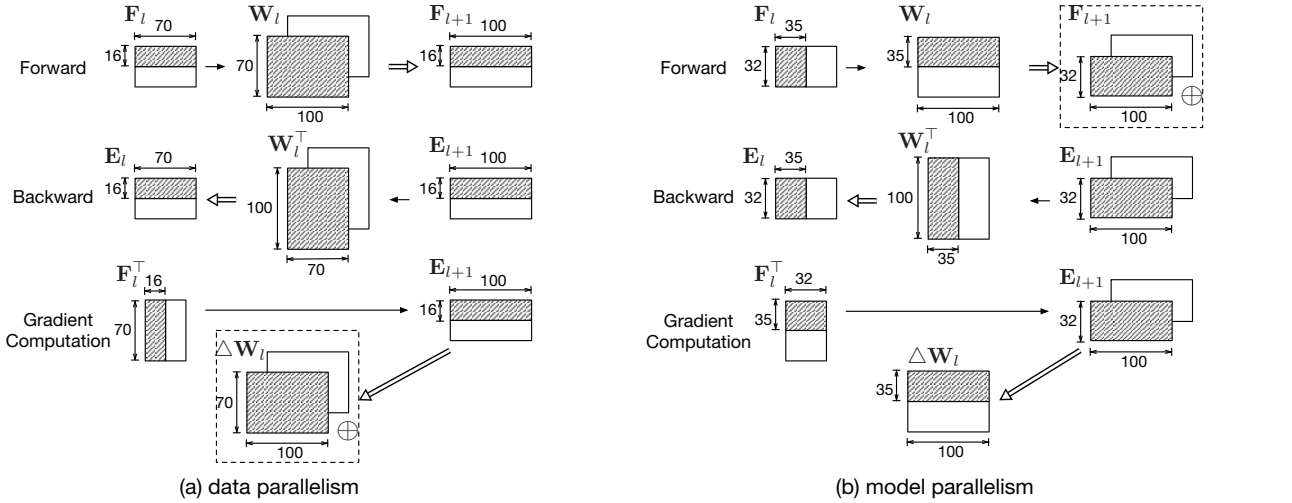


Fig. 9: Forward, Backward and Gradient Computation in (a) data parallelism and (b) model parallelism, and intra-layer communication is marked by a $\oplus$ [32].

TABLE I: Intra-layer communication in data parallelism and model parallelism [32].

data parallelism	model parallelism
$\mathbb{A}(\triangle \mathbf{W}_l)$	$\mathbb{A}(\mathbf{F}_{l+1})$

$\mathbf{W}_l$ and the feature map tensor $\mathbf{F}_l$ are used to calculate the error tensor in a previous layer as $\mathbf{E}_l = \left(\mathbf{E}_{l+1} \otimes \mathbf{W}_l^\top\right) \odot f'(\mathbf{F}_l)$, where $f'(\cdot)$ is the derivative function of $f(\cdot)$. In gradient computation, the partial derivatives to the weight is computed by $\triangle \mathbf{W}_l = \mathbf{F}_l^\top \otimes \mathbf{E}_{l+1}$. Since $f(\cdot)$, $f'(\cdot)$ and $\odot$ are element-wise operation and performed in-place, we focus on the three tensor multiplications, $\mathbf{F}_{l+1} = \mathbf{F}_l \otimes \mathbf{W}_l$, $\mathbf{E}_l = \mathbf{E}_{l+1} \otimes \mathbf{W}_l^\top$ and $\triangle \mathbf{W}_l = \mathbf{F}_l^\top \otimes \mathbf{E}_{l+1}$.

Fig. 9 shows the tensor shapes and computations in the basic data parallelism and model parallelism for two accelerators. The white tensors are assigned to one accelerator and the shadow tensors are assigned to the other. In data parallelism, we can see that there is no communication in forward and backward, but there is communication in gradient computation because each accelerator calculates the partial sum and need the partial sum from the other accelerator to get the final results. In model parallelism, we can see that there is no communication in backward and gradient computation, but there is communication in forward for the partial sum accumulation for the output feature map tensor. We call the above described data transfer as intra-layer communication. The intra-layer communication for data parallelism (dp) and model parallelism (mp) are listed in Table I.

There is another data transfer between tensors which we call it inter-layer communication. Since we have dp and mp as the two basic parallelism, we will have four inter-layer communication patterns, *i.e.*, dp-dp, dp-mp, mp-mp, mp-dp, as shown in Fig. 9. The inter-layer communication is caused by the remote accessing for a part of tensors that one accelerator requires to calculate feature map tensor for next layer of error tensor for previous layer but the accelerator does not have for this layer. The inter-layer communication for dp-dp, dp-mp, mp-mp and mp-dp are listed in Table II.

To minimize the total amount of communication, we can use a layer-wise dynamic programming method to search for

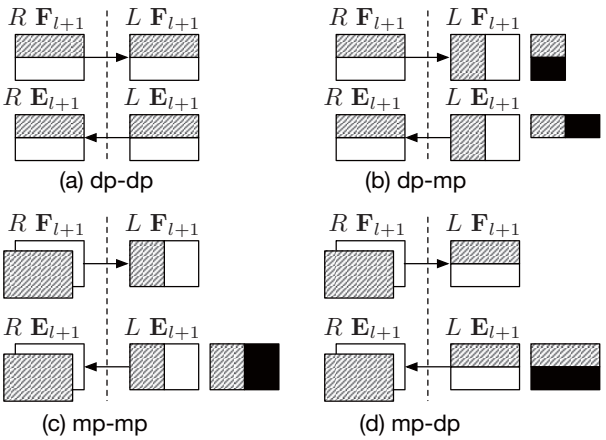


Fig. 10: Inter-layer communication for (a) dp-dp, (b) dp-mp, (c) mp-mp and (d) mp-dp [32].

TABLE II: Inter-layer communication for the transition of dp-dp, dp-mp, mp-mp and mp-dp [32].

dp-dp	0
dp-mp	$0.25\mathbb{A}(\mathbf{F}_{l+1}) + 0.25\mathbb{A}(\mathbf{E}_{l+1})$
mp-mp	$0.5\mathbb{A}(\mathbf{E}_{l+1})$
mp-dp	$0.5\mathbb{A}(\mathbf{E}_{l+1})$

the partitions for each layer. A hierarchical partition is used to recursively partition tensors to two sub-groups of accelerators until there is only one accelerator in a sub-group. We illustrate the hybrid parallelisms for VGG-A and AlexNet in Fig. 11. The data parallelism and model parallelism are interleaved across layers at one hierarchy and hierarchies of the same layer for a hybrid parallelism.

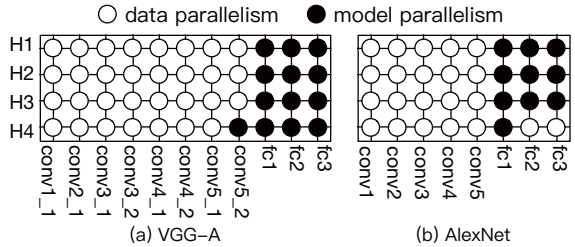


Fig. 11: The hybrid parallelism for (a) VGG-A and (b) AlexNet in [32].

V. Conclusion

In this paper, we systematically and constructively summarize the latest developments in the field of machine learning accelerator research. Previous works are classified into three categories based on the levels of parallelism in deep learning models being exploited in the proposed designs, namely processing engine-level parallelism, memory-level parallelism, and accelerator-level parallelism. The readers will understand the concept and design considerations of different parallelism levels, be able to identify and evaluate the effectiveness of different DNN hardware implementations, and apply these concepts to future DNN accelerator designs and accelerator designs in other areas such as graph processing [33], [34], [35] and genome sequencing [36], [37], [38].

Acknowledgments

This work is supported in part by DOE DE-SC0018064, NSF-1910299 and NSF CSR-1717885.

References

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, p. 436, 2015.

[2] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[3] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European conference on computer vision*, pp. 21–37, Springer, 2016.

[4] X. Liu, H. Yang, Z. Liu, L. Song, H. Li, and Y. Chen, "Dpatch: An adversarial patch attack on object detectors," *arXiv preprint arXiv:1806.02299*, 2018.

[5] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in neural information processing systems*, pp. 91–99, 2015.

[6] T. Ching *et al.*, "Opportunities and obstacles for deep learning in biology and medicine," *Journal of The Royal Society Interface*, vol. 15, no. 141, p. 20170387, 2018.

[7] O. Faust, Y. Hagiwara, T. J. Hong, O. S. Lih, and U. R. Acharya, "Deep learning for healthcare applications based on physiological signals: A review," *Computer methods and programs in biomedicine*, vol. 161, pp. 1–13, 2018.

[8] R. Miotto, F. Wang, S. Wang, X. Jiang, and J. T. Dudley, "Deep learning for healthcare: review, opportunities and challenges," *Briefings in bioinformatics*, vol. 19, no. 6, pp. 1236–1246, 2017.

[9] P. Baldi, P. Sadowski, and D. Whiteson, "Searching for exotic particles in high-energy physics with deep learning," *Nature communications*, vol. 5, p. 4308, 2014.

[10] G. B. Goh, N. O. Hodas, and A. Vishnu, "Deep learning for computational chemistry," *Journal of computational chemistry*, vol. 38, no. 16, pp. 1291–1307, 2017.

[11] L. Song, F. Chen, S. R. Young, C. D. Schuman, G. Perdue, and T. E. Potok, "Deep learning for vertex reconstruction of neutrino-nucleus interaction events with combined energy and time data," in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 3882–3886, IEEE, 2019.

[12] M. M. Waldrop, "The chips are down for moore's law," *Nature News*, vol. 530, no. 7589, p. 144, 2016.

[13] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," in *2011 38th Annual international symposium on computer architecture (ISCA)*, pp. 365–376, IEEE, 2011.

[14] M. Hu, H. Li, Q. Wu, and G. S. Rose, "Hardware realization of bsb recall function using memristor crossbar arrays," in *Proceedings of the 49th Annual Design Automation Conference*, pp. 498–503, ACM, 2012.

[15] H.-S. P. Wong, H.-Y. Lee, S. Yu, Y.-S. Chen, Y. Wu, P.-S. Chen, B. Lee, F. T. Chen, and M.-J. Tsai, "Metal–oxide rram," *Proceedings of the IEEE*, vol. 100, no. 6, pp. 1951–1970, 2012.

[16] M. Hu *et al.*, "Dot-product engine for neuromorphic computing: Programming 1t1m crossbar to accelerate matrix-vector multiplication," in *DAC*, p. 19, ACM, 2016.

[17] X. Liu *et al.*, "Reno: A high-efficient reconfigurable neuromorphic computing accelerator design," in *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 1–6, IEEE, 2015.

[18] X. Liu, M. Mao, B. Liu, B. Li, Y. Wang, H. Jiang, M. Barnell, Q. Wu, J. Yang, H. Li, *et al.*, "Harmonica: A framework of heterogeneous computing systems with memristor-based neuromorphic computing accelerators," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 63, no. 5, pp. 617–628, 2016.

[19] L. Song, X. Qian, H. Li, and Y. Chen, "Pipelayer: A pipelined reram-based accelerator for deep learning," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 541–552, IEEE, 2017.

[20] F. Chen, L. Song, and Y. Chen, "Regan: A pipelined reram-based accelerator for generative adversarial networks," in *2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 178–183, IEEE, 2018.

[21] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems 27*, pp. 2672–2680, Curran Associates, Inc., 2014.

[22] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *CoRR*, vol. abs/1511.06434, 2016.

[23] B. Li, L. Song, F. Chen, X. Qian, Y. Chen, and H. H. Li, "Reram-based accelerator for deep learning," in *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2018.

[24] Z. Fan, Z. Li, B. Li, Y. Chen, and H. H. Li, "Red: A reram-based deconvolution accelerator," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2019.

[25] F. Chen, L. Song, and Y. Chen, "Zara : A novel zero-free dataflow accelerator for generative adversarial networks in 3d reram," in *Proceedings of the 56th annual design automation conference*, ACM, 2019.

[26] F. Chen and H. Li, "Emat: An efficient multi-task architecture for transfer learning using reram," in *ICCAD*, 2018.

[27] M. Oquab, L. Bottou, I. Laptev, and J. Sivic, "Learning and transferring mid-level image representations using convolutional neural networks," in *CVPR*, June 2014.

[28] S. Han *et al.*, "Mcdnn: An approximation-based execution framework for deep stream processing under resource constraints," in *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '16, (New York, NY, USA), pp. 123–136, ACM, 2016.

[29] H. Ji, L. Song, L. Jiang, H. H. Li, and Y. Chen, "Recom: An efficient resistive accelerator for compressed deep neural networks," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 237–240, IEEE, 2018.

[30] Y. Wang, F. Chen, L. Song, R. Shi, H. Li, and Y. Chen, "Reboc: Accelerating block-circulant neural networks in reram," in *Design, Automation Test in Europe Conference Exhibition (DATE)*, 2020.

[31] C. Ding *et al.*, "CirCNN: Accelerating and Compressing Deep Neural Networks Using Block-Circulant Weight Matrices," in *MICRO*, 2017.

[32] L. Song, J. Mao, Y. Zhuo, X. Qian, H. Li, and Y. Chen, "Hypar: Towards hybrid parallelism for deep learning accelerator array," in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, IEEE, 2019.

[33] J. Ahn, S. Hong, S. Yoo, O. Mutlu, and K. Choi, "A scalable processing-in-memory accelerator for parallel graph processing," *ACM SIGARCH Computer Architecture News*, vol. 43, no. 3, pp. 105–117, 2016.

[34] G. Dai, T. Huang, Y. Wang, H. Yang, and J. Wawrzynek, "Graphsar: a sparsity-aware processing-in-memory architecture for large-scale graph processing on rerams," in *Proceedings of the 24th Asia and South Pacific Design Automation Conference*, pp. 120–126, ACM, 2019.

[35] L. Song, Y. Zhuo, X. Qian, H. Li, and Y. Chen, "Graphr: Accelerating graph processing using reram," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 531–543, IEEE, 2018.

[36] W. Huangfu, X. Li, S. Li, X. Hu, P. Gu, and Y. Xie, "Medal: Scalable dimm based near data processing accelerator for dna seeding algorithm," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 587–599, ACM, 2019.

[37] F. Zokaee, M. Zhang, and L. Jiang, "Finder: Accelerating fm-index-based exact pattern matching in genomic sequences through reram technology," in *2019 28th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pp. 284–295, IEEE, 2019.

[38] F. Chen, L. Song, H. Li, and Y. Chen, "Parc: A processing-in-cam architecture for genomic long read pairwise alignment using reram," in *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, ACM, 2020.