Performance Analysis of Cambricon MLU100

Jiansong Li[1,2*] and Zihan Jiang[1,2*]

[1] University of Chinese Academy of Sciences, Beijing
[2] Institute of Computing Technology, Chinese Academy of Sciences, Beijing
lijiansong@ict.ac.cn, jiangzihan@ict.ac.cn

Abstract. In recent years, domain-specific hardware has brought significant performance improvements in deep learning (DL). Many frequently-used optimization techniques, such as data parallelism, model parallelism, data pipeline, weights pruning and quantization have been proposed to accelerate the inference phase of DL workloads. However, there is still lack of a comparison of these optimization techniques to show their performance difference on dedicated accelerators. This paper evaluates these frequently-used optimization techniques on a commercial accelerator, namely Cambricon MLU100. Considering the requirement of accuracy of DL nature, our metric not only measures the inference throughput but also has an accuracy constraint. Based on our analysis methodology and performance numbers, we have some key observations and implications that are valuable for the future DL hardware and software co-design. Furthermore, we explore the upper bound of MLU100 inference performance under the standard ResNet-50 model and CIFAR-10 dataset.

Keywords: Deep Learning · Domain Specific Hardware · Performance Analysis

1 Introduction

Deep learning (DL) has revolutionized many challenge AI domains, such as image recognition [14, 23] and natural language processing [28, 29]. However, the large quantity of numerical operations and parameters induced by deep neural networks (DNNs) pose a signicant challenge to general-purpose processors. To keep pace with the growing computational demand in modern DL workloads, hardware specialization has become a popular way [15, 21, 3, 5, 25]. Therefore, many dedicated accelerators are gaining popularity for their performance efficiency. They have been deployed in edge devices, servers, and datacenters. For example, Huawei Meta10 and P20 cellphones integrated Cambricon-1A DL processor core [27]. Cambricon released MLU100 [1], which is a custom ASIC deployed in datacenter to accelerate the inference phase of morden DL workloads. And likewise, Google proposed Tensor Processing Unit to accelerate distributed machine learning [20].

* Equal contribution

Meanwhile, there are many frequently-used optimization techniques which enable the acceleration of the inference phase of modern DL workloads. These optimization techniques include but not limited to data parallelism, model parallelism, data pipeline, weights pruning and quantization [32, 9, 10, 17]. The performance variance among these optimization techniques poses a challenge for the future DL hardware and software co-design. Design or select appropriate optimization techniques on the dedicated DL accelerators is important and not easy. Moreover, there is still lack of a comparison of these optimization techniques to show their performance variance on dedicated accelerators. In this paper, we evaluate these frequently-used optimization techniques on a commercial DL accelerator—Cambricon MLU100. To systematically evaluate the platform, we sweep these frequently-used optimization techniques as hyperparameters. We take the standard ResNet-50 [13] model and CIFAR-10 [22] dataset as our benchmark, which is provided by BenchCouncil 2019 International AI system and Algorithm Challenges (Cambricon Track[3]). Our workload is from AIBench [7, 6], which is an AI benchmark for datacenter. The source code of AIBench is publicly available from http://www.benchcouncil.org/benchhub/AIBench (Sign up to get access). BenchCouncil organizes the international AI system challenges based on RISC-V [16], Cambricon chips [24, 30] and X86 platforms [2, 12, 4], and the international 3D face recognition algorithm challenges [31, 8]. BenchCouncil also provides AI benchmark for Edge [11], AIoT [26] and HPC [19]. Based on our analysis methodology and performance numbers, we conclude our observations and implications as following:

- Data pipeline and data parallelism significantly reduce the inference time while maintain the qualified accuracy.
- Compared with data parallelism, the impact of model parallelism on end to end inference throughput is not so significant.
- Weight pruning leads to a decline in accuracy although it may bring faster inference.
- High hardware throughput does not mean high end to end throughput.

Our observations and implications should help other researchers and practitioners to better the future DL hardware and software co-design. Furthermore, we explore the upper bound of MLU100 inference performance under the standard benchmark and reach an inference time of 384ms while preserving the target accuracy.

The rest of this paper is organized as follows: Sec. 2 introduces the background. Sec. 3 presents the evaluation methods and result is shown in Sec. 4. Sec. 5 concludes and discusses the future work.

[3] http://www.benchcouncil.org/competition/index.html

2 Background

2.1 Hardware Characteristics

Cambricon MLU100 [1] is a DL accelerator deployed in datacenter to accelerate the inference phase of modern DL workloads. Its ISA is based on Cambricon [25]. The general architecture of MLU100 is shown in Fig. 1. Cambricon MLU100 is based on the multi-core architecture. It includes four channels connected via a network on chip (NOC). Each channel contains one DDR and eight computational cores. For example, Channel0 contains one DDR memory controller (DDR0) and eight computational cores, namely C0, C1, ..., C7. *DDR* is responsible for the storage of DNN model, input and output of DL workloads. While those computational cores perform the execution of DNN computation tasks.

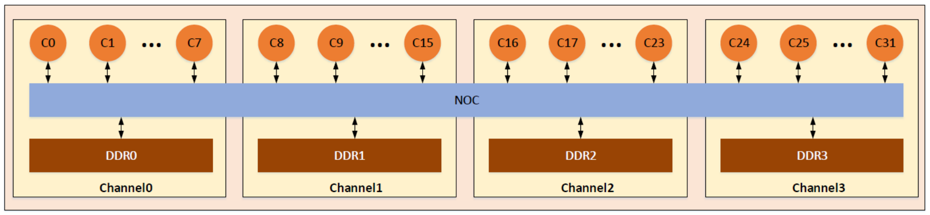


Fig. 1: Architectural information of Cambricon MLU100. Note that this picture is from Cambricon Caffe V0.9.7 documentation.

2.2 Software Stack

Fig. 2 shows the software stacks of Cambricon MLU100. As we all know, Caffe [18] is an open-sourced software framework used for DL training and inference. It is written in C++ and widely adopted in research experiments and industry deployments. Cambricon MLU100 provides Caffe as its high-level programming framework. Application programmers can simply deploy their applications via Cambricon Caffe. CNRT is the runtime toolkit of Cambricon MLU100. It provides some common low-level utility APIs, such as device and memory management, kernel launch, task queue scheduler and etc. CNML is a wrapper of CNRT. It provides some helper functions for DNN models' loading and execution and common highly-tuned DNN operators at MLU100, e.g., convolution and pooling operators. Driver and kernel is responsible for the handling of memory management and interrupts of MLU100.

2.3 Optimization Techniques

Cambricon Caffe provides some common utilities for optimization, such as data parallelism, model parallelism and data pipeline. These optimization techniques

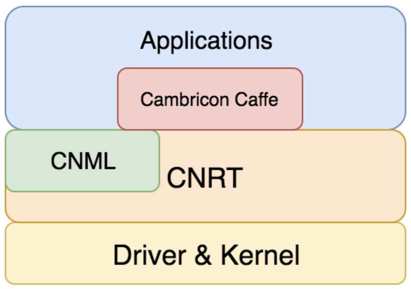


Fig. 2: Software Stacks of Cambricon MLU100. Note that this picture is from Cambricon Caffe V0.9.7 documentation.

usually improves the execution performance and preserves the final top-1 accuracy of DNN models during the inference phase. Besides, Cambricon Caffe supports weight pruning and quantization. While these optimization techniques usually have side effects over the final top-1 accuracy but improves the throughput. All these optimization techniques are not mutually exclusive.

Data Parallelism. In the inference phase of DL workloads, data parallelism means that given a CNN model, the input data is partitioned and assigned to different computational cores. As is shown in Fig. 3a, different cores have a complete copy of the DNN model. Each core simply gets a different part of the input data, and results from each core are somehow combined to get the final output. Data parallelism can greatly improve the throughout, since different parts of the input data can be executed concurrently.

Model Parallelism. As is shown in Fig. 3b, model parallelism means that different cores are responsible for the computations of different parts in a single network. For example, each layer in the neural network may be assigned to a different core. In the DL domain, we can take use of model parallelism by dividing a neural network into several subnets, and then putting each subnet into different cores of MLU100. Model parallelism can also improve the throughout, since for a single input, different parts of the DNN model can be executed concurrently.

Data Pipeline. In the inference phase of DL workloads, the input data flow will be fetched into host memory of CPUs from the disks, and then they will be uploaded into the device memory of MLU. Finally they will be fed to the computational cores of MLU. In this case, data pipeline can improve the workload balance of data prefetching, transfering and infeeding.

Weights Pruning and Quantization. As the large amounts of synaptic weights incur intensive computation and memory accesses in the inference phase of DL workloads, researchers have proposed a number of effective techniques to explore the sparsity of DNN, including weight pruning, model compression and quantization [32, 9, 10, 17]. Cambricon MLU100 trys to exploit the sparsity and irregularity of DNN models for the performance and power efficiency. It provides tools for weights pruning by setting the sparsity of the weights of input

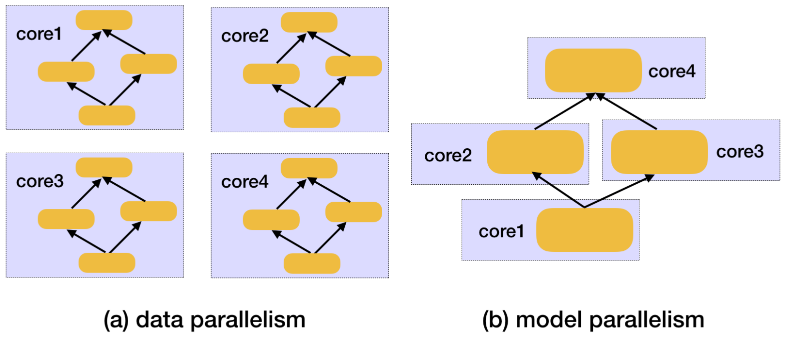


Fig. 3: Illustration of data parallelism and model parallelism.

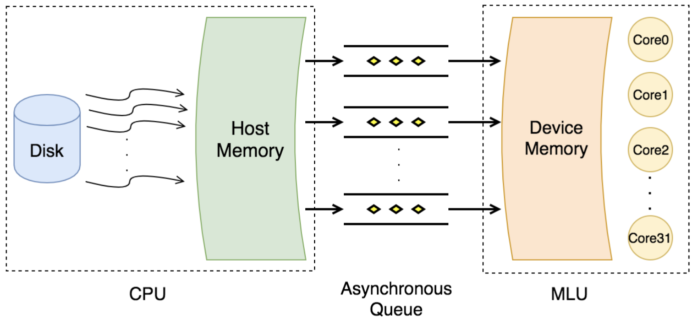


Fig. 4: Illustration of data pipeline. Note that to improve the throughput, we can launch multiple threads to read data from disks into the CPU memory and then dispatch the computational tasks into a queue that will be executed asynchronously. For those computational tasks within the same queue, they will be executed by their dispatching FIFO order. While those inter-queue tasks will be executed concurrently.

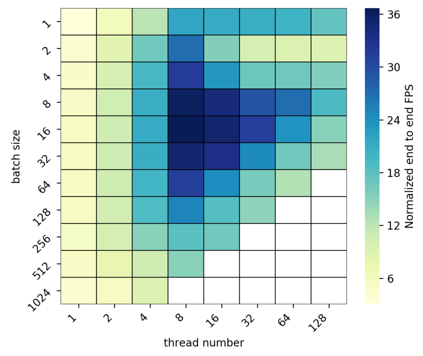
DNN model. Besides it provides tools to quantize the weights of DNN models into low-precision fixed-point numbers, e.g., INT8. We will discuss the effects of these optimization techniques over execution performance and top-1 accuracy in Sec. 4.

3 Evaluation Methods

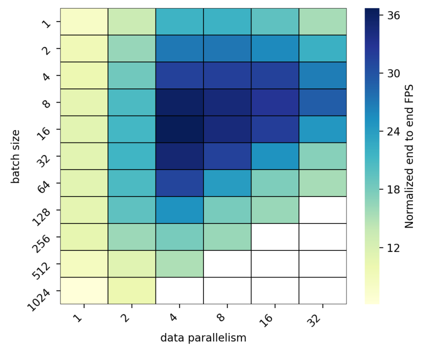
Our experiments run on a heterogeneous environment. The host CPU is a 2.10 GHz Intel(R) Xeon(R) CPU E5-2620 v4 machine and 16 cores/32 threads and 20 MB of L3 cache per socket and 128 GB of memory, running Ubuntu 16.04.10 LTS and GCC 5.4.0. The device accelerator is Cambricon MLU100 [1]. For Cambricon MLU100, its device memory is 8GB, whose bandwidth is 102.4 GB/s. The peak performance of MLU100 is 16 TFLOPS.

Table 1: The ranges of the hyperparameters chosen in this paper.

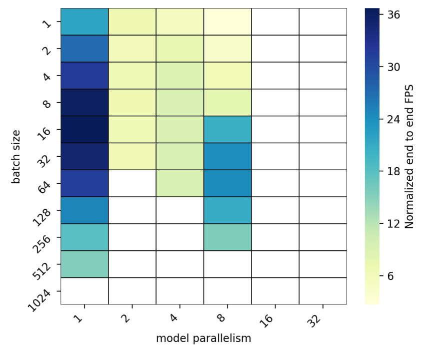
Variable	Batch Size	Data Parallelism	Model Parallelism	Thread Number	Sparsity
Min	1	1	1	1	0.10
Max	1024	32	32	128	0.90
Inc	*2	*2	*2	*2	+0.01



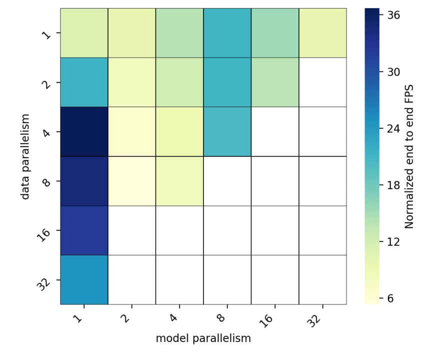
(a) Batch size vs thread number.



(b) Batch size vs data parallelism.



(c) Batch size vs model parallelism.



(d) Data parallelism vs model parallelism.

Fig. 5: The effects of batch size, thread number, data parallelism and model parallelism over execution performance. Note: the blank space means NaN. The FPS values are normalized to the speedup ratio over the end to end case where batch size, data parallelism, model parallelism and thread number are 1, 1, 1, 1 respectively.

Table 1 summarizes the hyperparameters chosen in this paper and how they are swept. Cambricon Caffe provides tools to set the *data parallelism* and *model parallelism* for the inference task. *Thread number* is the number of threads to be launched to read data from disks. *Batch size* is a hyperparameter that defines the number of image samples to be loaded in current iteration of inference tasks. *Sparsity* is a hyperparameter that specifies the degree of zeros of the weights. For example, if its value is 0.3, that means 30% of the weights data will be zero. We evaluate the performance of Cambricon MLU100 under the standard ResNet50 [13] model and CIFAR-10 [22] datasets. The programming framework in this paper is Cambricon Caffe. For all workloads, we run 20 times and calculate the average.

4 Performance Numbers

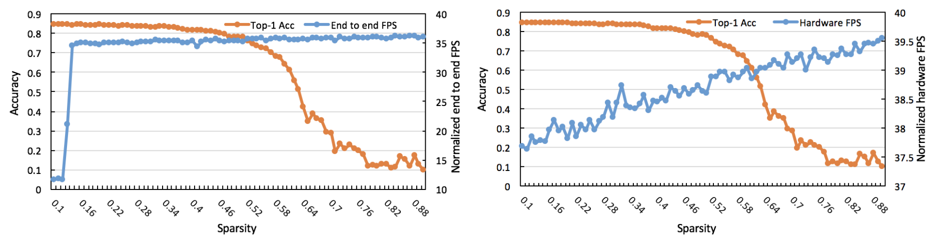
All these hyperparameters in Table 1 are not mutually exclusive, we group these hyperparameters based on whether they affect the final top-1 accuracy. In this section, we discuss the effects of these hyperparameters over the final accuracy and execution performance.

4.1 Optimizations Preserving Accuracy

Hyperparameters like *batch size, thread number, data parallelism and model parallelism* usually have no side effects over the final top-1 accuracy. By sweeping the first four hyperparameters in Table 1, we find that the final top-1 accuracy of ResNet-50 over CIFAR-10 datasets at MLU100 preserves at 84.39%. In the terms of end to end FPS, the best configuration for the first four hyperparameters in Table 1 is 16, 4, 1, 8. To demonstrate the effects of *batch size, thread number, data parallelism and model parallelism* over execution performance, we choose this best configuration as guideline. For each case in Fig. 5, we sweep the corresponding two hyperparameters by fixing the other twos. As we can see from Fig. 5a and Fig. 5b, data pipeline and data parallelism significantly improve the end to end throughput. Meanwhile, we can see from Fig. 5c and Fig. 5d that the impact of model parallelism on end to end throughput is not so significant. For ResNet-50, best model parallelism is 1 in the terms of end to end throughput.

4.2 Optimizations Affecting Accuracy

As we mentioned in Sec. 4.1, the best configuration for the first four hyperparameters in Table 1 is 16, 4, 1 and 8 separately in the terms of end to end FPS. To demonstrate the effects of weight pruning over execution performance and top-1 accuracy, we choose this best configuration as guideline. We sweep the *sparsity* hyperparameter from 0.1 to 0.9 by fixing the other fours to the best configuration. As we can see from Fig. 6, weights pruning significantly affects the end to end and hardware throughput. In Fig. 6a, the hardware FPS increases when the input weight sparsity increases, since higher sparsity means more zeros in the weights data and higher throughput of the device accelerators. However, as is shown in Fig. 6b, with the increase of weights sparsity, the end to end FPS improvement slows down. That means high hardware throughput does not mean high end to end throughput, because of the load imbalance of data



(a) Sparsity over accuracy and end to end FPS.

(b) Sparsity over accuracy and hardware FPS.

Fig. 6: The effects of weight pruning over execution performance and top-1 accuracy. Note that the x-axis is the sparsity of weights; the left y-axis is the global top-1 accuracy; the right y-axis is normalized FPS (frame per second), which can be treated as a proxy of FLOPS. The FPS values are normalized to the speedup ratio over the end to end case where batch size, data parallelism, model parallelism and thread number are 1, 1, 1, 1 respectively.

feeding between host CPUs and device accelerators. Besides, the top-1 accuracy drops significantly with the increase of weights sparsity. In the real production environment, although weights pruning brings higher end to end throughput, a very low accuracy may not be acceptable. For the quantization optimization techiniques, the top-1 accuracy of MLU100 maintains at a qualified level.

5 Conclusion and Future Work

In this paper, we investigate many frequently-used optimization techniques including but not limited to *data parallelism, model parallelism, data pipeline, weights pruning* and *quantization*. And compare these optimization techniques to show their performance difference at throughput and top-1 accuracy on Cambricon MLU100. Based on our analysis methodology and performance numbers, we conclude our observations and implications which will help to better the future DL hardware and software co-design. As future work, we are planning to evaluate more DNN models on more DL accelerators.

Acknowledgement. This work is partially supported by the National Key R&D Program of China(under Grant No. 2017YFB1003103).

References

1. Cambricon: Cambricon MLU100. http://www.cambricon.com/index.php?c=page&id=20
2. Chen, M., Chen, T., Chen, Q.: Hygon-alswr: An efficient implementation of the alswr algorithm on hygon x86 cpus. In: International Symposium on Benchmarking, Measuring and Optimization (Bench'19). Springer (2019)

3. 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. In: ACM Sigplan Notices. vol. 49, pp. 269–284. ACM (2014)
4. Deng, W., Wang, P., Wang, J., Li, C., Guo, M.: Psl: Exploiting parallelism, sparsity and locality to accelerate matrix factorization on x86 platforms. In: International Symposium on Benchmarking, Measuring and Optimization (Bench'19). Springer (2019)
5. 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)
6. Gao, W., Luo, C., Wang, L., Xiong, X., Chen, J., Hao, T., Jiang, Z., Fan, F., Du, M., Huang, Y., Zhang, F., Wen, X., Zheng, C., He, X., Dai, J., Ye, H., Cao, Z., Jia, Z., Zhan, K., Tang, H., Zheng, D., Xie, B., Li, W., Wang, X., Zhan, J.: AIBench: Towards Scalable and Comprehensive Datacenter AI Benchmarking. In: International Symposium on Benchmarking, Measuring and Optimization. pp. 3–9. Springer (2018)
7. Gao, W., Tang, F., Wang, L., Zhan, J., Lan, C., Luo, C., Huang, Y., Zheng, C., Dai, J., Cao, Z., et al.: AIBench: An Industry Standard Internet Service AI Benchmark Suite. arXiv preprint arXiv:1908.08998 (2019)
8. Gong, T., Huiqian, N.: An implementation of resnet on the classification of rgb-d images. In: International Symposium on Benchmarking, Measuring and Optimization (Bench'19). Springer (2019)
9. 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. International Conference on Computer Architecture (ISCA) (2016)
10. Han, S., Mao, H., Dally, W.J.: Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. International Conference on Learning Representations (ICLR) (2016)
11. Hao, T., Huang, Y., Wen, X., Gao, W., Zhang, F., Zheng, C., Wang, L., Ye, H., Hwang, K., Ren, Z., et al.: Edge AIBench: Towards Comprehensive End-to-end Edge Computing Benchmarking. In: International Symposium on Benchmarking, Measuring and Optimization. pp. 23–30. Springer (2018)
12. Hao, T., Zheng, Z.: The implementation and optimization of matrix decomposition based collaborative filtering task on the hygon x86 platform. In: International Symposium on Benchmarking, Measuring and Optimization (Bench'19). Springer (2019)
13. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. CoRR **abs/1512.03385** (2015), http://arxiv.org/abs/1512.03385
14. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016)
15. Hennessy, J.L., Patterson, D.A.: A new golden age for computer architecture. Commun. ACM **62**(2), 48–60 (2019)
16. Hou, P., Yu, J., Miao, Y., Tai, Y., Wu, Y., Zhao, C.: Rvtensor: A light-weight neural network inference framework based on the risc-v architecture. In: International Symposium on Benchmarking, Measuring and Optimization (Bench'19). Springer (2019)
17. Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., Bengio, Y.: Quantized neural networks: Training neural networks with low precision weights and activations. J. Mach. Learn. Res. **18**(1), 6869–6898 (Jan 2017), http://dl.acm.org/citation.cfm?id=3122009.3242044

18. 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)
19. Jiang, Z., Gao, W., Wang, L., Xiong, X., Zhang, Y., Wen, X., Luo, C., Ye, H., Lu, X., Zhang, Y., et al.: HPC AI500: A Benchmark Suite for HPC AI Systems. In: International Symposium on Benchmarking, Measuring and Optimization. pp. 10–22. Springer (2018)
20. Jouppi, N.P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., Bates, S., Bhatia, S., Boden, N., Borchers, A., Boyle, R., Cantin, P.l., Chao, C., Clark, C., Coriell, J., Daley, M., Dau, M., Dean, J., Gelb, B., Ghaemmaghami, T.V., Gottipati, R., Gulland, W., Hagmann, R., Ho, C.R., Hogberg, D., Hu, J., Hundt, R., Hurt, D., Ibarz, J., Jaffey, A., Jaworski, A., Kaplan, A., Khaitan, H., Killebrew, D., Koch, A., Kumar, N., Lacy, S., Laudon, J., Law, J., Le, D., Leary, C., Liu, Z., Lucke, K., Lundin, A., MacKean, G., Maggiore, A., Mahony, M., Miller, K., Nagarajan, R., Narayanaswami, R., Ni, R., Nix, K., Norrie, T., Omernick, M., Penukonda, N., Phelps, A., Ross, J., Ross, M., Salek, A., Samadiani, E., Severn, C., Sizikov, G., Snelham, M., Souter, J., Steinberg, D., Swing, A., Tan, M., Thorson, G., Tian, B., Toma, H., Tuttle, E., Vasudevan, V., Walter, R., Wang, W., Wilcox, E., Yoon, D.H.: In-datacenter performance analysis of a tensor processing unit. In: Proceedings of the 44th Annual International Symposium on Computer Architecture. pp. 1–12. ISCA '17, ACM, New York, NY, USA (2017). https://doi.org/10.1145/3079856.3080246, http://doi.acm.org/10.1145/3079856.3080246
21. Jouppi, N.P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., Bates, S., Bhatia, S., Boden, N., Borchers, A., et al.: In-datacenter performance analysis of a tensor processing unit. In: 2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA). pp. 1–12. IEEE (2017)
22. Krizhevsky, A.: The CIFAR-10 dataset. http://www.cs.toronto.edu/ kriz/cifar.html
23. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems. pp. 1097–1105 (2012)
24. Li, G., Wang, X., Ma, X., Liu, L., Feng, X.: XDN: Towards Efficient Inference of Residual Neural Networks on Cambricon Chips. In: International Symposium on Benchmarking, Measuring and Optimization (Bench'19). Springer (2019)
25. Liu, S., Du, Z., Tao, J., Han, D., Luo, T., Xie, Y., Chen, Y., Chen, T.: Cambricon: An instruction set architecture for neural networks. In: ACM SIGARCH Computer Architecture News. vol. 44, pp. 393–405. IEEE Press (2016)
26. Luo, C., Zhang, F., Huang, C., Xiong, X., Chen, J., Wang, L., Gao, W., Ye, H., Wu, T., Zhou, R., et al.: AIoT Bench: Towards Comprehensive Benchmarking Mobile and Embedded Device Intelligence. In: International Symposium on Benchmarking, Measuring and Optimization. pp. 31–35. Springer (2018)
27. Medium: Huawei 7nm Kirin 810 Beats Snapdragon 855 and Kirin 980 on AI Benchmark Test. https://medium.com/syncedreview/huawei-7nm-kirin-810-beats-snapdragon-855-and-kirin-980-on-ai-benchmark-test-af31996fb10e
28. Sutskever, I., Vinyals, O., Le, Q.V.: Sequence to sequence learning with neural networks. In: Advances in neural information processing systems. pp. 3104–3112 (2014)
29. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. In: Advances in neural information processing systems. pp. 5998–6008 (2017)

30. Wang, Y., Zeng, C., Li, C.: Exploring the Performance Bound of Cambricon Accelerator in End-to-End Inference Scenario. In: International Symposium on Benchmarking, Measuring and Optimization (Bench'19). Springer (2019)
31. Xiong, X., Wen, X., Huang, C.: Improving rgb-d face recognition via transfer learning from a pretrained 2d network. In: International Symposium on Benchmarking, Measuring and Optimization (Bench'19). Springer (2019)
32. 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. pp. 20:1–20:12. MICRO-49, IEEE Press, Piscataway, NJ, USA (2016), http://dl.acm.org/citation.cfm?id=3195638.3195662