RVTensor: A light-weight neural network inference framework based on the RISC-V architecture

Pengpeng Hou[1,2], Jiageng Yu[1*], Yuxia Miao[1], Yang Tai[1], Yanjun Wu[1], Chen Zhao[1]

[1] The Institute of Software, Chinese Academy of Sciences
[2] University of Chinese Academy of Sciences
{pengpeng, jiageng08, yuxia, taiyang, yanjun,zhaochen}@iscas.ac.cn

Abstract. The open-source instruction set architecture RISC-V has developed rapidly in recent years, and its combination mode of multiple sub-instruction sets has attracted the attention of IoT vendors. However, research on the IoT scenario inference framework based on the RISC-V architecture is rare. Popular frameworks such as MXNet, TensorFlow, and Caffe are based on the X86 and ARM architectures, and they are not optimized for the IoT scenarios. We propose RVTensor that a light-weight neural network inference framework based on the RISC-V architecture. RVTensor is based on the SERVE.r platform and is optimized for resource-poor scenarios. Our experiments demonstrate that the accuracy of RVTensor and the Keras is the same.

Keywords: RISC-V, deep learning, resnet20, SERVE.r

1 Introduction

This paper is for 2019 BenchCouncil International Artificial Intelligence System Challenges. The Challenges has four tracks: International AI System Challenge based on RISC-V(we participate in this), International AI System Challenge based on Cambricon Chip[1][2], International AI System Challenge based on X86 Platform [3][4][5], International 3D Face Recognition Algorithm Challenge[6][7].

The RISC-V instruction set architecture[8] consists of a basic instruction set and multiple extended instruction sets, which is more flexible than the X86 and ARM architectures. The flexible combination mode of the RISC-V architecture is suitable for the IoT domain with various requirements. The IoT vendors can personally combine and customize the instruction set according to the specific scenarios, so the RISC-V architecture has a good application prospect in the IoT field.

Few deep learning inference frameworks support the RISC-V architecture. TensorFlow[9], MXNet[10], Caffe[11], and PaddlePaddle[12] are based on the X86 and GPU architectures, lacking support for RISC-V architectures, and these frameworks often run on resource-rich servers and have few optimizations for the IoT domain.

*Corresponding author: Jiageng Yu

RVTensor: A light-weight neural network inference framework based on the RISC-V architecture

We propose a deep learning inference framework RVTensor that is optimized for resource-poor application scenarios based on the RISC-V architecture. It has few third-party dependent libraries, small memory requirements, and small executable files. We have evaluated RVTensor by the resnet20 model, and find that the accuracy of its results is the same as the Keras[13].

2 Overall architecture

The overview of RVTensor architecture is shown in Fig.1, it consists of four parts: model analysis, op operators, construction calculation graph, and execution calculation graph.

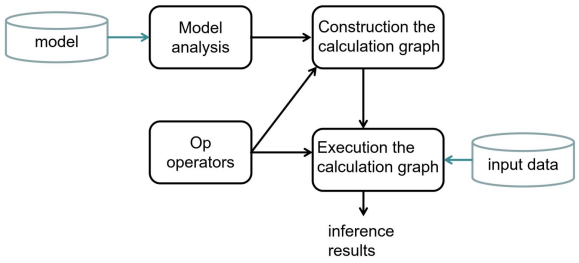


Fig.1. Overview of RVTensor architecture

Model analysis. It mainly parses model files such as *.pb*, and extracts information such as operator operations and weight data.

Op operators. It mainly includes the implementation of each operator, including *conv, add, active, pooling, fc* [14] and other operations. These operators are the basic computing unit of the neural network execution.

Construction the calculation graph. It builds a calculation graph based on the model analysis and the op operator modules. In the calculation graph, the order between the operators, the dependencies between the weights and the operators are clarified.

Execution the calculation graph. It obtains the inference results based on the input data (such as image data) and the calculation graph.

3 Optimization skills

RVTensor is optimized for resource-poor scenarios by reducing dependencies on third-party libraries and increasing memory utilization. To reduce the dependence on third-party libraries, we propose many light-weight API interfaces, e.g., we re-implement thread function APIs. To improve memory utilization, we reuse the memory.

Re-implement thread API. Pthread [15] is a popular open-source thread library, which provides APIs such as thread creation, thread waiting, thread exit, thread lock. However, RVTensor mainly involves two thread functions: thread creation and thread waiting, most functions in the Pthread library are not involved. Therefore the Pthread is quite heavy for RVTensor, and we re-implemented the involved functions based on the RISC-V architecture instead of using the Pthread. The way we implement the functions is

shown in Figure 2. We propose the thread creation interface in thread.c. First, we allocate the stack memory for the new thread and then invoke the *clone* function to create the thread. The *clone* function is implemented in the clone.c file. In the clone.c, we first parse the parameters and then call the interface in the clone.s to create threads based on the parameters. The clone.s is an assembly file that wraps the *sys_clone* system call to really create the thread. Compared to the Pthread, the new thread API is more lightweight and more suitable for IoT scenarios.

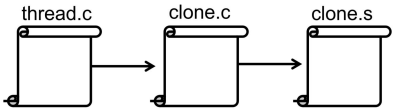
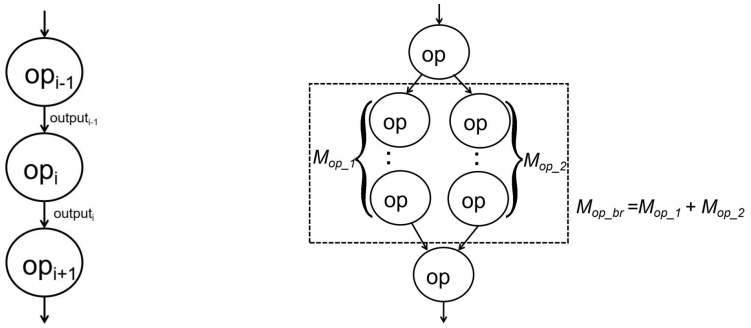


Fig. 2. Thread function call procedure

Memory reuse. The hardware of the IoT scenario generally has a small memory capacity. If the memory optimization is lacking, the inference system is prone to have OOM errors[16]. We propose a memory reuse strategy to save the memory. The classic segment of the calculation graph is shown in Figure 3(a), the memory space for the current op_i operator is necessary, and the space for the op_{i-1} and op_{i+1} do not need to be allocated immediately. So we create a global memory block for the current op operator to reuse.



(a) segment of the calculation graph (b) two sub-branches in the calculation graph

Fig. 3. Execution branch of the calculation graph

The memory space M_{op} required for the op operator is calculated as follows:

$$M_{op} = \sum_{i=0}^{n} (M_{ii} + M_{ik} + M_{ib}) + M_o \qquad (1)$$

Where M_{ii} represents the size of the *i*-th input data, M_{ik} represents the kernel size of the *i*-th input data, M_{ib} represents the bias size of the *i*-th input data, and M_o represents the output size of the op operator.

The global memory block M_{pool} shared by each operator is calculated as follows:

$$M_{pool} = \max(M_{op1}, M_{op2}, ...M_{opn}) \qquad (2)$$

If the graph has multiple branches in a stage, as shown in Fig3(b), we treat the sub-branches as a atomic operation, take the sum of the M_{op_i} of each sub-branch as the M_{op_br} that represents the memory space needed in this stage, as shown in the following formula:

$$M_{op\_br} = \sum_{i=0}^{n} M_{op\_i} \qquad (3)$$

Where M_{op_i} represents the max M_{op} in the *i*-th sub-branch.

The memory reuse strategy saves memory space while reducing the number of memory allocations.

4 Evaluation

We evaluate RVTensor based on the SERVE.r platform[17], the test model is the resnet20[18], and the data set is the cifar10[19], involving 10,000 images. Our experiment refers to the benchmark for image classification in AIBench[20][21][22]. AIBench source code is publicly available from http://www.benchcouncil.org/benchhub/AIBench/ (Sign up to get access).

Accuracy. We use the Keras as the baseline. After analyzing the test results, the accuracy of RVTensor on *Top1* and *Top5* is 77% and 98%, respectively, which is the same as the Keras'. It should be noted that the Keras cannot be installed and run on the SERVE.r platform due to the memory limitation, so its accurate evaluation is based on the X86 platform.

Table 1. Accuracy based on the resnet20 model

	Top1	Top5
RVTensor	77%	98%
Keras	77%	98%

Performance. The average time to process each image is 13.51 seconds.

Execution file size. The executable file size of RVTensor is 193KB.

5 Future work

The performance of RVTensor is still low, and we will improve it in the following ways:

- Memory optimization. Due to the limited memory of the SERVE.r, there will be memory swapping in and out, which will cause the time overhead of some op operators to increase significantly and reduce the performance. We will try to solve this issue.

- Optimization for sparse convolution. The *relu* activation will result in an ample number of zeros in the input data, these zeros occupy a large space and cause the convolution operation to be inefficient. We will optimize the sparse convolution operation.

- The V instruction set adaptation. We will try to re-implement the op operator based on the V instruction set, improving the operator execution efficiency.

- Model pruning. We will compress the model parameters through pruning techniques to make them more suitable for IoT scenes.

6 Summary

We propose a deep learning inference framework RVTensor for the IoT scenario based on the RISC-V architecture. We reduce the dependency on third-party libraries by implementing light-weight API interface, and save memory space during the op operator calculation process through the memory reuse strategy. Our experiments demonstrate that the accuracy of RVTensor and the Keras is the same, and RVTensor's executable file is quite small.

References

[1]. Li, Guangli and Wang, Xueying and Ma, Xiu and Liu, Lei and Feng, Xiaobing(2019). XDN: Towards Efficient Inference of Residual Neural Networks on Cambricon Chips. International Symposium on Benchmarking, Measuring and Optimization (Bench'19).

[2]. Li, Jiansong and Jiang, Zihan(2019). Performance Analysis of Cambricon MLU100.International Symposium on Benchmarking, Measuring and Optimization (Bench'19).

[3]. Deng, Weixin and Wang, Pengyu and Wang, Jing and Li, Chao and Guo, Minyi(2019). PSL: Exploiting Parallelism, Sparsity and Locality to Accelerate Matrix Factorization on x86 Platforms. International Symposium on Benchmarking, Measuring and Optimization (Bench'19).

[4]. Hao, Tianshu and Zheng, Ziping(2019).The Implementation and Optimization of Matrix Decomposition Based Collaborative Filtering Task on x86 Platform. International Symposium on Benchmarking, Measuring and Optimization (Bench'19).

[5]. Chen, Maosen and Chen, Tun and Chen, Qianyun(2019).An Efficient Implementation of the ALS-WR Algorithm on x86 CPUs.International Symposium on Benchmarking, Measuring and Optimization (Bench'19).

[6]. Xiong, Xingwang and Wen, Xu and Huang, Cheng(2019). Improving RGB-D face recognition via transfer learning from a pretrained 2D network. International Symposium on Benchmarking, Measuring and Optimization (Bench'19).

[7]. Gong, Tongyan and Niu Huiqian(2019).An Implementation of ResNet on the Classification of RGB-D Images. International Symposium on Benchmarking, Measuring and Optimization (Bench'19).

[8]. RISC-V. https://riscv.org/.

[9]. Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. TensorFlow: a system for large-scale machine learning. In Proceedings of the 12th USENIX conference on Operating Systems Design and Implementation (OSDI'16). USENIX Association, Berkeley, CA, USA, 265-283.

[10]. Chen, T., Li, M., Li, Y., Lin, M., Wang, N., Wang, M., ... & Zhang, Z. (2015). Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. arXiv preprint arXiv:1512.01274.

[11]. Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., ... & Darrell, T. (2014, November). Caffe: Convolutional architecture for fast feature embedding. In Proceedings of the 22nd ACM international conference on Multimedia (pp. 675-678). ACM.

[12]. PaddlePaddle. https://www.paddlepaddle.org.cn/

[13]. Keras. https://keras.io/

[14]. Long, J., Shelhamer, E., & Darrell, T. (2015). Fully convolutional networks for semantic segmentation. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 3431-3440).

[15]. Pthread. http://man7.org/linux/man-pages/man7/pthreads.7.html

[16]. OOM. https://en.wikipedia.org/wiki/Out_of_memory

[17]. SERVE.r https://github.com/ict-accel-team/SERVE.r

[18]. He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).

[19]. Cifar. https://www.cs.toronto.edu/~kriz/cifar.html

[20]. Gao, Wanling & Tang, Fei & Wang, Lei & Zhan, Jianfeng & Lan, Chunxin & Luo, Chunjie & Huang, Yunyou & Zheng, Chen & Dai, Jiahui & Cao, Zheng & Zheng, Daoyi & Tang, Haoning & Zhan, Kunlin & Wang, Biao & Kong, Defei & Wu, Tong & Yu, Minghe & Tan, Chongkang & Li, Huan & Ye, Hainan. (2019). AIBench: An Industry Standard Internet Service AI Benchmark Suite. arXiv preprint arXiv:1908.08998.

[21]. Gao, Wanling & Luo, Chunjie & Wang, Lei & Xiong, Xingwang & Chen, Jianan & Hao, Tianshu & Jiang, Zihan & Fan, Fanda & Du, Mengjia & Huang, Yunyou & Zhang, Fan & Wen, Xu & Zheng, Chen & He, Xiwen & Dai, Jiahui & Ye, Hainan & Cao, Zheng & Jia, Zhen & Zhan, Kent & Zhan, Jianfeng. (2018). AIBench: Towards Scalable and Comprehensive Datacenter AI Benchmarking. 2018 BenchCouncil International Symposium on Benchmarking, Measuring and Optimizing (Bench18) (pp. 3-9).

[22]. AIBench. http://www.benchcouncil.org/AIBench/index.html