Leveraging Subgraph Extraction for Performance Portable Programming Frameworks on DL Accelerators

Xiao Zhang[1,2,3](✉), Huiying Lan[1], and Tian Zhi[1]

[1] Intelligent Processor Research Center, Institute of Computing Technology (ICT), CAS, Beijing, China
zhangxiao@ict.ac.cn
[2] University of Chinese Academy of Sciences (UCAS), Beijing, China
[3] Cambricon Tech. Ltd., Beijing, China

Abstract. Deep learning framework plays an important role in connecting hardware platform and algorithm. In recent years, some domain-specific deep learning accelerators with better performance and energy efficiency were proposed by researchers. However, current frameworks lack enough considerations about how to better support the possible new features brought by accelerators. In this paper, we propose to build a performance portable programming framework with subgraph extraction. The intuition is that increasing ratio of optimizations are taken from the top-level framework to the low-level software stack of accelerator. In response to this development trend, framework needs to pay more attention to the splitting strategy of computation graph for the heterogeneous computation.

1 Introduction

In recent years, we have witnessed many significant breakthroughs of deep learning algorithm in a multitude of domains. This superior accuracy, however, comes at the cost of high computational complexity. Researchers try to design more efficient architectures based on the features of deep learning algorithm and get some promising results [3–5,7–10]. These results show that domain-specific accelerators outstand in both speed and energy efficiency compared to traditional solutions.

On the other hand, in order to explore and deploy deep learning algorithm conveniently, both academia and industry have developed several deep learning frameworks, such as MXNet [2], TensorFlow [1] and Caffe [6]. Those frameworks automatically optimize the computation flow, generate high-performance kernels and schedule kernels in parallel if possible.

However, there is a gap between emerging DL accelerators and existing programming frameworks. In order to run deep learning algorithm with the highest performance, some accelerators and its software stacks have tried to break the wall and search optimal solution in a large space. Unfortunately, current deep learning frameworks only provide limited adaptions for this new feature.

F. Zhang et al. (Eds.): NPC 2018, LNCS 11276, pp. 179–184, 2018.
https://doi.org/10.1007/978-3-030-05677-3_21

2 Motivation

2.1 DLA and Graph Fusion

We designed and implemented a deep learning accelerator and its software stack, and we call the accelerator DLA in following sections. The design of DLA is concluded from multiple deep learning accelerators, including NVidia DLA, DaDianNao [4] and TPU. There are multiple cores in DLA. Each core in DLA can complete a computation task independently, which makes it actually a parallel model with shared global memory.

Compared to traditional limited method that fusing some specific sequence composed of element-wise operators issued by framework, software stack of DLA offers a more radical solution. It optimizes and fuses the total graph (see the Fig. 1). This strategy has several benefits. First, the experts developed lower stack can give better solution because they know more about hardware architecture. Also, fusing a large graph into a single node greatly saves the kernel launch cost, which is important for inference task.

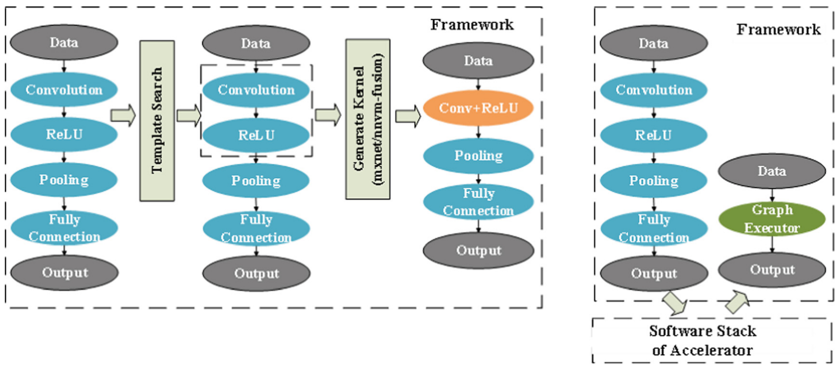

Fig. 1. In left part, the framework searches limited templates and generates new kernels to replace them. In right part, optimization stack of accelerator receives the whole graph, optimizes and generates a new executor back to framework.

2.2 Heterogeneous Computation

Heterogeneous computation is unavoidable for DLA and other accelerators. Some operators in new algorithms are hard to parallelize or to abstract to the tensor operators offered by accelerators, and the frequency of embedding accelerator in mobile device might be reduced to save energy. As a result, assigning some parts on CPU might bring better total performance. Thus, before we use lower software stack to optimize graph, we need to extract a subgraph composed by operators assigned on DLA. In other words, framework should have a clever split strategy and method to extract appropriate subgraph from the original deep networks.

3 Subgraph Extraction

When we try to extract a subgraph based on whether each operator is well-supported by accelerator, the direct intuition is to make it a maximum connected convex subgraph. Connectivity guarantees data relation between operators which is necessary for most optimizing methods. Maximum grants the largest searching space and reduces kernel launch overheads. Convexity is used as a constraint to avoid circle which leads to dead lock when scheduling. A subgraph **S** of a directed acyclic graph **G** is convex if and only if there is no directed path between two vertices of **S** which contains an arch not in **S** (see the Fig. 2).

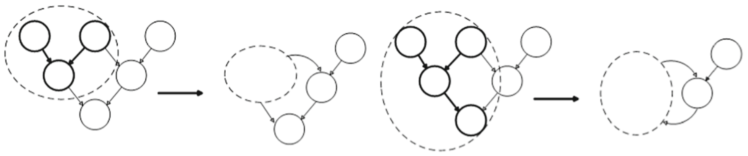

Fig. 2. Example of convex and non-convex subgraph.

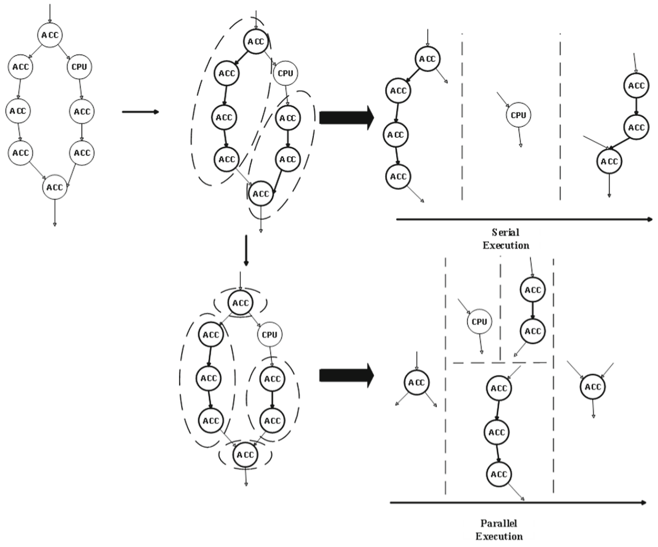

Fig. 3. Post-prune strategy. The ACC node represents operator assigned on accelerator, and the CPU node represents operator assigned on CPU.

Merging a large subgraph into a single node helps the corresponding computation to run faster, however, it may hinder scheduler to get maximum parallelism in some case. As Fig. 3 shows, the fused graph must wait for all its input to be ready even though some inputs are not necessary at the early stage of its computation. Similarly, although not all the outputs of a subgraph are generated at the final stage, all descendants must keep waiting until computation of total subgraph finishes. So, we append a post-prune process to split each subgraph into smaller parts, each of which has only one input and output operator.

4 Evaluation

The experiment platform is DLA, a multi-core deep learning accelerator as we mentioned before. We first evaluate the performance before and after the graph fusion to demonstrate the validation of graph fusion. As shown in Fig. 4, performance of all six entire-network benchmarks are improved, which achieves a speedup of 1.18× on average compared with the baseline, which we do not implement the graph fusion. Specifically, the improvement of ResNet34 and ResNet50 is clearly higher than other four networks.

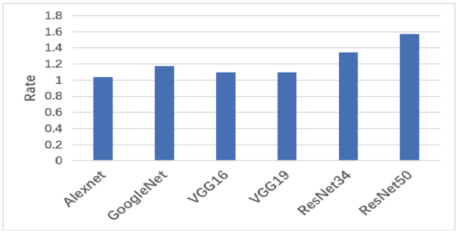

Fig. 4. Relative speedup of graph.

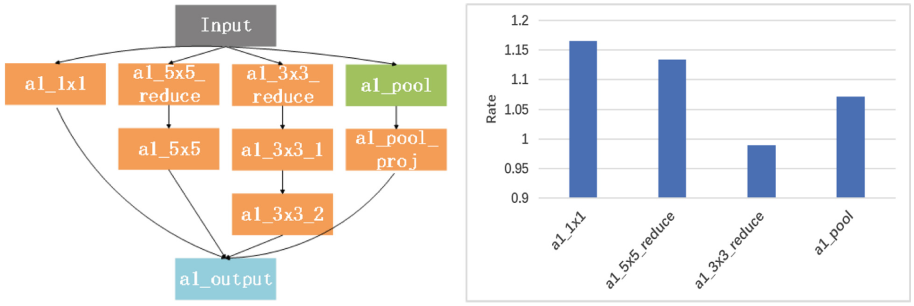

Fig. 5. Left figure shows the structure of the inception-v3 block. Right figure shows speedup after the post-prune strategy. Horizontal axis label represents part of the block assigned to CPU

Then we evaluate the speedup of the post prune process. We use the intuitive maximum connected convex subgraph extraction strategy as the baseline.

In order to accurately evaluate the prune strategy, we choose a basic block of operators with multiple branches from inception-v3 networks for its enough braches. To trigger subgraph extraction, we separately assign operators on different branch to CPU and evaluate the speedup. As the result shown in the Fig. 5, except for assigning operator on the critical path to CPU, performance of the other three heterogeneous computation get a speedup of 1.1× on average, which is an obvious improvement.

5 Conclusion

In this paper, we propose a performance portable programming framework. The key motivation is that framework needs a subgraph extraction strategy to better balance schedule parallelism and fusion efficiency. We implement such a framework by migrating MXNet. This strategy is designed to cooperate framework with lower software stack in heterogeneous computation task, because none of them can complete the whole task independently. This strategy can be used in a wider field if accelerators choose to take over framework to optimize the computation graph by themselves.

Acknowledgment. This work is partially supported by the National Key Research and Development Program of China (under Grant 2017YFA0700902, 2017YFB1003101), the NSF of China (under Grants 6147239, 61432016, 61473275, 61522211, 61532016, 61521092, 61502446, 61672491, 61602441, 61602446, 61732002, 61702478), the 973 Program of China (under Grant 2015CB358800), National Science and Technology Major Project (2018ZX01031102) and Strategic Priority Research Program of Chinese Academy of Sciences (XDBS01050200).

References

1. Abadi, M., et al.: Tensorflow: a system for large-scale machine learning (2016)
2. Chen, T., et al.: MXNet: a flexible and efficient machine learning library for heterogeneous distributed systems. Statistics (2015)
3. Chen, T., et al.: DianNao: a small-footprint high-throughput accelerator for ubiquitous machine-learning. In: Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 269–284. ACM (2014)
4. Chen, Y., et al.: DadianNao: a machine-learning supercomputer. In: IEEE/ACM International Symposium on Microarchitecture, pp. 609–622 (2014)
5. Du, Z., et al.: ShiDianNao. ACM SIGARCH Comput. Arch. News **43**(3), 92–104 (2015)
6. Jia, Y., et al.: Caffe: convolutional architecture for fast feature embedding. arXiv preprint arXiv:1408.5093 (2014)
7. Liu, D., et al.: PuDianNao: a polyvalent machine learning accelerator. In: Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 369–381 (2015)
8. Liu, S., et al.: Cambricon: an instruction set architecture for neural networks. In: Proceedings of the 43rd International Symposium on Computer Architecture, pp. 393–405. IEEE Press (2016)

9. Reagen, B., et al.: Minerva: enabling low-power, highly-accurate deep neural network accelerators. In: ACM SIGARCH Computer Architecture News, vol. 44, pp. 267–278. IEEE Press (2016)
10. Zhang, S., et al.: Cambricon-X: an accelerator for sparse neural networks. In: IEEE/ACM International Symposium on Microarchitecture, pp. 1–12 (2016)