Glow vs TVM - The Art of Compiler Optimization for Deep Learning Frameworks

CSE 598 Data Intensive Systems for Machine Learning, Spring 2020

Parth Doshi
pdoshi4@asu.edu

1. TVM

1.1. Summary

TVM is an AI-tuned automated End-to-End Optimizing compiler which is open source and can be used with CPUs, GPUs, and different specialized hardware accelerators in the deep learning framework. The idea is to bridge the gap between productivity-focused deep learning frameworks, and efficiency-oriented hardware back-end systems. [2]

By exposing the graph-level and operator-level optimizations, the main aim of TVM is to provide performance portability for deep learning workloads amongst diverse hardware back-ends. Optimization challenges like high-level operator fusion, memory latency hiding and mapping to arbitrary hardware primitives are solved by TVM. Low-level programs are automatically optimized according to the hardware characteristics by employing a novel, learning-based cost modeling method, thereby enabling rapid exploration of code optimizations. [2]

TVM includes end-to-end automated optimization, which is a highly specialized labor-intensive task.[1] It also supports multiple deployment back-ends in languages such as C++, Java and Python. [2]

1.2. Motivation

In order to design and develop a deep learning application, one can choose from the diverse set of deep learning frameworks like PyTorch, TensorFlow, Caffe, Theano, etc. The choice of hardware to train and deploy these Deep Learning models is huge. However, due to this diversity of options, we have several challenges as an application developer. [3]

Some of the challenges of having diverse Deep Learning frameworks and hardware platforms are:

- There is a difficulty to switch from one framework to another due to the differences among the front-end interfaces and the back-end implementations. Each framework is well-suited for different situations. [3]
- To guarantee performance on various hardware systems, multiple back-ends need to be maintained by the framework developers.[3]
- Multiple frameworks need to be supported in the new chips designed by the chip vendors. Since the workloads in different framework are represented and executed in unique ways, a simple single operation like a convolution will have to be defined in respective way. [3]

1.3. Strengths

- Heavy run-time environments like TensorFlow run-time environment can be replaced by lightweight run-time environment like LLVM, thereby releasing load from the host machine. [3]
- Due to the AutoTVM module, it provides better performance than top Machine Learning frameworks such as TensorFlow as optimizations are automatically tuned according to the host machine back-end. [3]
- TVM can optimize DL workloads over multiple platforms while it can speedup relatively compared to mobile NN framework and mobile libraries with less time for ResNet-18 and MobileNet. It can also support depthwise-convolution, low-precision operations while optimize for new specialized accelerators. [1]

1.4. Weaknesses

- The best kernels in the 10 generations, 100 population (2000 trails per operator) are offered by TensorComprehension(TC). 2D convolution is heavily optimized by cuDNN. However, for most of the layers, TVM generates better GPU kernels. [2]

- TC requires almost no manual experience or scheduling optimization, compared with manual implementations in TensorFlow(TF) and the optimization experience needed by TVM. [2]

- The throughput achieved by TVM is not equivalent to that of manually optimized implementations. [2]

2. Glow

2.1. Summary

Glow (Graph Lowering) is a community driven pragmatic approach to compilation that helps deep learning framework generate highly optimized codes for multiple targets. A two-phase strongly-typed intermediate representation is lowered from the traditional neural network data-flow by Glow. Domain-specific optimizations are performed due to this high-level intermediate representation. Memory-related optimizations, such as instruction scheduling, static memory allocation and copy elimination are performed with lower-level instruction-based address-only intermediate representation.[4] At the lowest level, the optimizer performs machine-specific code generation to take advantage of specialized hardware features.

2.2. Motivation

Whole-graph optimizations are needed for efficient execution of the neural networks on DSAs. Traditionally, the nodes in the graph are executed one by one in the machine learning frameworks, which is unfortunately inefficient, even on traditional processors. [4]

Due to this, compilers are used to execute graph code more efficiently in the machine learning frameworks. We can see an increasing importance of neural networks currently and in the future while the need for efficient energy consumption in data centers and mobile devices is ever growing. Also, with the design principles of domain-specific architectures, it is of paramount important for a machine learning framework to focus on providing attractive programming models on top of a layer that integrates compilers for many different back-ends. [4]

Glow focuses on the software stack's lower parts. Machine learning frameworks are provided with a low-level graph and a code generator for their neural networks. Graph lowering is the main technique that the compiler uses for generating efficient code. The aim is not to replace the machine learning high-level graph but provide a useful compiler toolkit allowing hardware developers to focus on implementing efficient acceleration hardware with differing capabilities. [4]

2.3. Strengths

- In order to make efficient use of the hardware execution units, the compiler creates a schedule that hides the memory operations latency due to the instruction-based lower-level IR. [4]

- By eliminating the need to implement all the operators on all targets, a lowering phase is present in Glow to support a high number of input operators as well as a large number of hardware targets.

- The intuition behind the lowering phase is to reduce the input space in such a way that allows new hardware back-ends to focus on a small number of linear algebra primitives. [4]

- By observing the execution during the inference, the numeric range is estimated for each stage of the network. Glow converts these floating-point based networks into signed 8-bit integers networks because arithmetic using small integers is memory and performance efficient. [4]

2.4. Weaknesses

- Profile-guided quantization is implemented in Glow for the conversion of floating-point operands to 8-bit integers. But for estimation of values, the training-based quantization is more robust and better. [4]

- As the compiler needs to schedule memory efficiently, Glow currently unrolls RNN networks, thereby making it difficult to implement early termination. [4]

- LLVM is used to optimize and produce machine code by Glow, however Glow uses LLVM to optimize and produce machine code. But, LLVM works with far fewer languages and generates binaries for fewer platforms. [4]

References

[1] T. Chang. Tvm: an ai-tuning ai-compiler, 2019. 1

[2] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, M. Cowan, H. Shen, L. Wang, Y. Hu, L. Ceze, and et al. Tvm: An automated end-to-end optimizing compiler for deep learning. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, OSDI'18, page 579–594, USA, 2018. USENIX Association. 1, 2

[3] O. Foundation. Tvm: A deep learning compiler stack, 2018. 1

[4] N. Rotem, J. Fix, S. Abdulrasool, G. Catron, S. Deng, R. Dzhabarov, N. Gibson, J. Hegeman, M. Lele, R. Levenstein, J. Montgomery, B. Maher, S. Nadathur, J. Olesen, J. Park, A. Rakhov, M. Smelyanskiy, and M. Wang. Glow: Graph lowering compiler techniques for neural networks, 2018. 2, 3