Paper Review

Tony Wang
Technical University Munich

1 Introduction

Tianqi Chen, lead author of the paper holds a BS and MS from the Shanghai Jiao Tong Universit, a PhD from the University of Washington and is currently an Assistant Professor at Carnegie Mellon University. At the time the article was published in 2018, Chen was still a PhD student at the University of Washington.

Deep learning computation requirements are increasing each year; therefore, it is imperative to fetch every bit of performance, and optimize everywhere. Thus, new hardware need to get published to the market, which also creates a problem. Many of these novel hardware have different architectures. But current deep learning frameworks are only optimized on server-class GPUs and not for these specialized hardware, since they are based on operator libraries. which are hand-written optimizations by software engineers. They delegate target-specific optimizations to highly engineered and vendor-specific operator libraries, which is requires experience and can cost a lot of development time. Therefore deep learning frameworks perform worse on many specialized hardware, for they are not optimized for them. If you want to deploy your model on a new hardware that your framework does not support, you might end up with bad performance. To target this problem, Chen et al. proposed a novel end-to-end compiler called TVM (Tensor Virtual Machine), which can address a wide range of different hardware back-ends.

2 Summary

Chen et al. use in TVM an idea that dates way back to 1997 - that is optimizing the code with machine-learning. Since then, there has been many attempts at using machine-learning (ML) for compilers. The work of Chen et al. uses this approach to solve two fundamental problems in deep learning in one fell swoop. It can not only target a wide range of hardware back-ends, it also generates optimized code for each of the hardware. This process takes place in many stages, which are explained in a well-structured manner throughout the paper. TVM takes a model from different frameworks (e.g. TensorFlow or PyTorch) and converts it into a computation graph for high-level graph optimizations, such as operator fusion and data layout transformation and creates an optimized graph. Then for each operator in the graph, the operator-levle optimization module, the heart of the compiler, generates the target specific optimized code. In this module the schedule and execution details will be determined for the specific hardware-backend by the ML-based cost model. The resulting code can then be deployed on different modules, like LLVM IR, CUDA, or OpenCL. TVM also implements RPC-based remote function call support, which is crucial for embedded devices. At last the authors evaluate their new system on a server-class GPU, embedded GPU and CPU, and a custom-made FPGA-accelerator with real world workloads, yielding promising results, by surpassing the state-of-the-art rivals in every benchmark.

3 Review

Deep learning frameworks depend on creating computation graphs, thus prominent deep learning frameworks such as PyTorch and TensorFlow make use of graph-level optimizations as well. In this area other deep learning frameworks are very similar to TVM. For instance, TensorFLow uses Grapper as graph optimization system, using the same graph optimizations such as Constant folding optimizer, or Layout optimizer. Chen et al. respond to this by emphasising that, in contrary to other deep learning frameworks, TVM does not rely on hand-built kernel libraries and can therefore target a much wider area of hardware, since it is not feasible to handshape the hand-written libraries for every hardware. Chan et al. then go into detail how operator fusion and data layout transformation work. However, I would have loved to see a more in-depth comparison between these graph-level optimizations between these frameworks. Since they all share similar tasks, and therefore implement similar operations, it would be interesting to see if the numbers of possible operations match with each other, or whether some graph-level optimizing op-

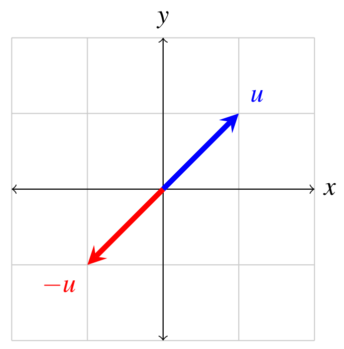

Figure 1: Text size inside figure should be as big as caption's text. Text size inside figure should be as big as caption's text. Text size inside figure should be as big as caption's text. Text size inside figure should be as big as caption's text. Text size inside figure should be as big as caption's text.

erations perform better than other frameworks and vice versa.

Here's a typical reference to a floating figure: Figure 1. Floats should usually be placed where latex wants then. Figure1 is centered, and has a caption that instructs you to make sure that the size of the text within the figures that you use is as big as (or bigger than) the size of the text in the caption of the figures. Please do. Really.

In our case, we've explicitly drawn the figure inlined in latex, to allow this tex file to cleanly compile. But usually, your figures will reside in some file.pdf, and you'd include them in your document with, say, \includegraphics.

Lists are sometimes quite handy. If you want to itemize things, feel free:

fread a function that reads from a `stream` into the array `ptr` at most `nobj` objects of size `size`, returning returns the number of objects read.

Fred a person's name, e.g., there once was a dude named Fred who separated usenix.sty from this file to allow for easy inclusion.

The noindent at the start of this paragraph in its tex version makes it clear that it's a continuation of the preceding paragraph, as opposed to a new paragraph in its own right.

3.1 LaTeX-ing Your TeX File

People often use `pdflatex` these days for creating pdf-s from tex files via the shell. And `bibtex`, of course. Works for us.

Acknowledgments

The USENIX latex style is old and very tired, which is why there's no \acks command for you to use when acknowledging. Sorry.

Availability

USENIX program committees give extra points to submissions that are backed by artifacts that are publicly available. If you made your code or data available, it's worth mentioning this fact in a dedicated section.