A load-aware scheduler for large-scale neural network autotuning

PROJECT THESIS II / T2000

for the study program
Computer Science

at the
Baden-Wuerttemberg Cooperative State University Stuttgart

by
Dominik Stiller

Submission Date	September 2019
Project Period	18 Weeks
Company	Hewlett Packard Enterprise
Corporate Supervisor	Junguk Cho
University Supervisor	Prof. Dr. Bernd Schwinn
Matriculation Number, Course	4369179, TINF17A

Declaration of Authorship

I hereby declare that the thesis submitted with the title *A load-aware scheduler for large-scale neural network autotuning* is my own unaided work. All direct or indirect sources used are acknowledged as references.

Neither this nor a similar work has been presented to an examination committee or published.

Sindelfingen	September 3$^{\text{rd}}$, 2019	
Place	Date	Dominik Stiller

Abstract

Real-time computer vision applications with deep learning-based inference require hardware-specific optimization to meet stringent performance requirements. However, this approach requires vendor-specific libraries developed by experts for some particular hardware, limiting the set of supported devices and hindering innovation. The deep learning compiler stack TVM is developed to address these problems. TVM generates the optimal low-level implementation for a certain target device based on a high-level input model using machine learning in a process called autotuning.

In this paper, we first explore the capabilities and limitations of TVM's autotuning implementation. Then, we develop a scheduler to orchestrate multiple, parallel autotuning jobs on shared computation resources such as CPUs and GPUs, allowing us to minimize resource idle time and job interference. Finally, we reflect our design choices and compare the efficiency of our approach with the default, scheduler-less design.

Contents

List of Figures

List of Tables

List of Source Codes

1 Introduction

motivation: AI is increasing in popularity AI is used in many different areas they arent experts Existing products for easier setup and deployment of training and inference infrastructure

1.1 Motivation

Often real-time necessary (examples) requires low inference time use of specific accelerator devices Generic models perform poorly because they dont make full use of accelerator capabilities Model needs to be attuned to accelerator Requires deep knowledge -> not easy for non-expert users

1.2 Problem

required: automated inference performance optimization (autotuning) so model performance can keep up with application demands imagine autotuning as a service where users can submit their trained model and receive an optimized version according to SLA Would make autotuning available for a wide audience on a large scale Only autotuning implementation in deep learning compiler stack TVM (find evidence), what we will be using throughout this project Only supports single jobs

1.3 Scope

In this paper, we enable large scale autotuning be developing a scheduler to orchestrate multiple jobs Design and create a working proof-of-concept implementation

Research question: What improvements of efficiency in terms of - autotuning speed - resulting inference performance and - resource utilization does a scheduler have, compared to the default design?

dont improve actual autotuning, but propositions are made in future work dont write actual autotuning as a service product, would require user interface

2 Deep Learning

Machine learning Consists of training and inference

2.1 Artificial Neural Networks

general structure with layers layers: fully-connected, batch norm basically a function mapping from input (image, text, data) to result (e.g. classification) trend towards more layers to create more complex function which are more powerful many layers: deep neural network

2.2 Convolutional Neural Networks

more layers: pooling and convolutions important for computer vision

2.3 Computation Graph Representation

in essence just a lot of calculations calculations can be represented in computation graph graph describes which calculations need to be done and stays the same tensor operators implement convolutions, pooling and need to be optimized for accelerator hardware

3 Inference Optimization

additionally to traditional training and inference deep learning workflow, we introduce inference performance optimization to meet real-time requirements include graphic showing train-inference vs train-optimize-inference

3.1 Tensor Operator Optimization

to optimize for minimal inference time of whole network, we need to optimize every layer/tensor operator many possible implementations only few optimal ones for target device focus on convolutions, as opposed to dense (why?) one layer corresponds to one tensor operator with a specific shape

3.2 Manual Optimization

state of the art cuDNN and TensorRT, taken as baseline requires deep knowledge of target device limitations - no support for new devices - no support for unconventional shapes - no support for new graph-level optimizations

3.3 Automated Optimization / Autotuning

using machine learning vendor-agnostic define autotuning job, task describe autotuning process schedules as abstraction with knobs

4 TVM

as far as we know, TVM is only framework that implements autotuning To create a scheduler, we examined TVM (commit id) with a few modifications to support measurements (check what else we changed)

written in Python and C++, interoperating import from many frontends, compilation for many backends has own graph-level and tensor operator-level representation calls target-specific compiler

4.0.1 RPC Architecture

allows autotuning logic to run on powerful server, but profiling to happen on target device

4.1 SimpleTVM

created wrapper for simpler usage of TVM expose easy, chainable interface forms base of scheduler internally Created Docker container to be able to easily deploy TVM with all dependencies on any server

4.1.1 Automated Benchmarking Framework

superb enable automated testing of different configurations to be able to run multiple configurations without human intervention

4.2 Experiments

Explore TVM behavior in different configurations of concurrency and resource sharing Evaluation in Jupyter notebooks

4.3 Issues with current autotuning design

During experiments we found some issues Since only a single job is supported, AaaS is not possible efficiently We cant just run two jobs since they might interfere, as previous experiments show Also, lots of idle time due to synchronous design Dont want idle time because edge resources are limited (define edge) Enabling controlled parallel autotuning is necessary to solve those problems necessitates central scheduler that orchestrates all jobs

5 Autotuning Scheduler

objectives: Be able to run an arbitrary number of autotuning jobs while 1. maximizing inference performance 2. minimizing autotuning time 3. maximizing resource usage in order of priority

5.1 Design

general idea: interleave stages while keeping dependencies and preventing interference include graphics from poster

to keep scheduler algorithm simple, we designed it to be agnostic of stages scheduler needs to know - knows which job will use which resource - knows which resource is currently available we call this load-aware theoretically, could work for any application that supports this interface

client provides interface between autotuning logic and scheduler

show abstract scheduler and client interface show scheduling pseudocode

allows for variable strategies to compare different designs

since only proof-of-concept, very specific and non-flexible/fault-tolerant

5.2 Implementation

Since TVM only provides a python interface, we are using python 3.5

5.2.1 RPC

We want clients to live in different docker containers, possibly physical servers (why?) requires RPC infrastructure consisting of scheduler and clients different from TVM RPC infrastructure clients register to scheduler describe endpoints

5.3 Challenges

evaluation of design choices takes long because autotuning is a slow process initially wanted to run scheduler and clients in one multi-threaded process without RPC to get results quickly not possible due to python global interpreter lock what else?

6 Evaluation

Comparison of interleaved design vs synchronous and sequential in terms of autotuning time and inference time hardware and network specifications

Evaluation only with limited set of hardware and models, general statement requires more experiments Especially running jobs in parallel that vary significantly in complexity need to be examined

7 Conclusion

Describe results

7.1 Future Work

make into mature product

Predictive scheduler using times for task to make scheduling more intelligent, as idea for future work Keep trained model and update it every n new entries to skip loading history time for every task Check currently known best configurations and see if SLA is already met before actually starting autotuning Automatically set up autotuning infrastructure

Bibliography

[1] Test, “Hello world,” 2019.