EECS 583 Research Paper Presentation

Hustin Cao, Aylin Gunal, Shinka Mori

Towards Neural Architecture-Aware Exploration Of Compiler Optimizations in a Deep Learning {Graph} Compiler

Gaurav Verma
gaurav.verma@stonybrook.edu
Stony Brook University
Stony Brook, NY, USA

Swetang Finviya
sfinviya@cs.stonybrook.edu
Stony Brook University
Stony Brook, NY, USA

Abid M. Malik
amalik@bnl.gov
Brookhaven National Laboratory
Upton, NY, USA

Murali Emani
memani@anl.gov
Argonne National Laboratory
Lemont, IL, USA

Barbara Chapman
barbara.chapman@stonybrook.edu
Stony Brook University
Stony Brook, NY, USA

Published in ACM, 2022

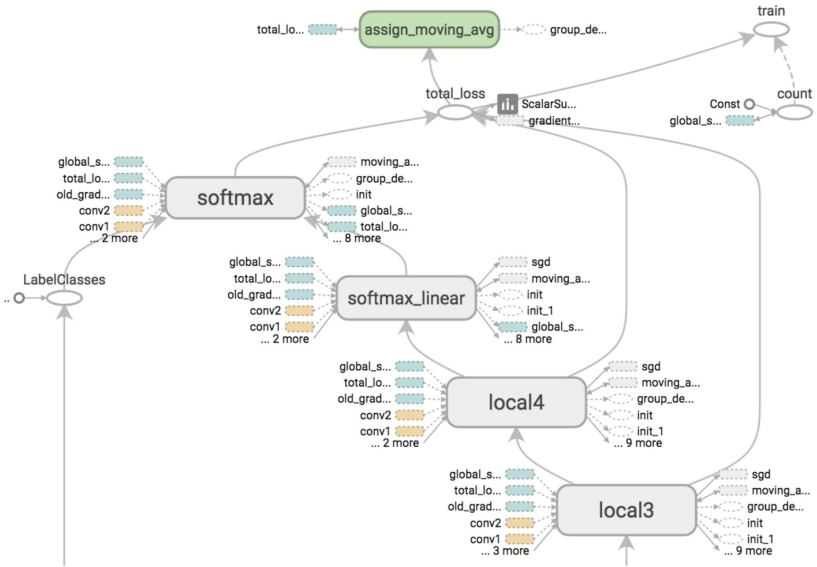
https://bpb-us-e1.wpmucdn.com/you.stonybrook.edu/dist/6/1671/files/2022/07/Towards-Neural-Architecture-Aware-Exploration-Of-Compiler-Optimizations-in-a-Deep-Learning-Graph-Compiler.pdf

Background

- Development of graph-based deep learning compilers leads to massive search space for optimizations targeting computation graphs generated from Deep Neural Network code
  - Depending on what optimizations are applied, the intermediate representation (IR) differs significantly, affecting overall performance
- Prior work has investigated use of machine learning to narrow down the search space for (1) pass selection; (2) pass ordering
- **Authors propose (manual) categorization of passes as relevant to different neural network architectures**
  - Experiments in TVM with pre-existing TVM passes

Computation Graph Optimization

Sample TensorFlow computation graph
representation of a neural network



https://cs230.stanford.edu/section/5/

- Computation graph is a type of IR; popular software to create them include TensorFlow, Pytorch, MXNet
  - Idea is to model dependencies between operations

Methodology – Neural Network Architectures

Analyze architectural differences in deep neural networks to employ different optimization passes. Researchers chose four different architectures.

1. ResNet - (ResNet50, 50-layer deep CNN)
    a. Task: Image classification // Dataset: ImageNet
2. MobileNet (MobileNetV2, 53-layer deep CNN)
    a. Task: Image classification // Dataset: COCO
3. SSD (SSD-ResNet50, Single Shot CNN)
    a. Task: Object detection // Dataset: CIFAR-10
4. BERT (bi-directional encoder transformer)
    a. Task: Question-Answering task // Dataset: SQuAD 2.0

Methodology – Compiler Optimization Analysis

From observations, the researchers found that **two or more optimization levels can produce the exact same IR** and **an optimization level may contain a set of passes that does not affect the IR.**

Therefore, only consider passes that would will affect the IR of the target NN.

1. Baseline Passes (just setting OPT_LEVEL=3)
2. ResNet Class
3. MobileNet Class
4. SSD Class
5. BERT Class
6. Additional Passes (Certain passes are dependent on user's (programmer's) intent on the code)

Pass Categorization

- Passes are categorized based on whether or not they are directly relevant to a particular Deep NN architecture
  - Decided based on number of network layers, types of tensor operations, and their order
- **Core idea: prune the search space by removing passes that are irrelevant to the given architecture**

Pass	Description	Category
AlterOpLayout	Used for computing convolution in custom layouts or other general weight pre-transformation.	BL; RN; MN; SSD; BR
AnnotateSpans	Annotate a program with span information	AD
BatchingOps	Batching parallel operators into one for Conv2D, Dense and BatchMatmul	BR
CanonicalizeCast	Canonicalize cast expressions to make operator fusion more efficient.	BL; RN
CanonicalizeOps	Canonicalize special operators to basic operators	BL
CombineParallelConv2D	Combine multiple conv2d operators into one	RN
CombineParallelDense	Combine multiple dense operators into one	RN
ConvertLayout	Alternate the layouts of operators	BL
DeadCodeElimination	Remove expressions that do not have any usage	RN; MN; SSD
DefuseOps	Inverse operation of fusion pass.	BL
DynamicToStatic	Convert dynamic operations to static if possible	AD
EliminateCommonSubexpr	Eliminate common subexpressions	BL; RN; MN
FakeQuantizationToInteger	Takes fake quantized graphs and convert them to actual integer operations	AD
FastMath	Convert expensive non linear functions to their fast but approximate counterparts	MN; BR; SSD
FirstOrderGradient	Transform all global functions in the module to return the original result and the gradients of the inputs.	MN
FoldConstant	Fold the constant expressions in a Relay program	RN; MN; SSD
FoldExplicitPadding	Find explict padding before an operator that supports implicit padding and fuses them.	RN; SSD
ForwardFoldScaleAxis	Fold the scaling of axis into weights of conv2d/dense	BR
FuseOps	Fuse operators in an expression to a larger operator	RN; MN; SSD; BR
MergeComposite	Merge multiple operators into a single composite relay function	RN; MN; SSD
PartitionGraph	Partition a Relay program into regions that can be executed on different backends	AD
RemoveUnusedFunctions	Remove unused global relay functions in a relay module	SSD
SimplifyExpr	Simplify the Relay expression, including merging consecutive reshapes.	RN; MN; SSD
SimplifyFCTranspose	Simplify the transpose operation on a dense layer	MN; BR
SimplifyInference	Simplify the data-flow graph for inference phase.	RN; MN; SSD
ToANormalForm	Turn Graph Normal Form expression into A Normal Form Expression	AD
ToGraphNormalForm	Turn a normal form into graph normal form.	RN; MN
ToMixedPrecision	Automatic mixed precision rewriter	AD

*BL: baseline optimization passes having OPT_LEVEL=3; RN: ResNet Class; MN: MobileNet Class; SSD: SSD_ResNet Class; BR: BERT Class; AD: Additional optimizations

Pass selection and ordering for SSD_ResNet50 in PyTorch

BL = baseline, AS-x = architecture-informed, PO-x = random selection from search space

Key	
BL	AlterOpLayout, CanonicalizeCast, CanonicalizeOps, ConvertLayout, DefuseOps, EliminateCommonSubexpr
AS-0	AlterOpLayout, FuseOps, SimplifyExpr, FoldConstant, DeadCodeElimination, MergeComposite, FastMath, RemoveUnusedFunctions
AS-1	SimplifyExpr, FuseOps, AlterOpLayout, MergeComposite, FastMath, DeadCodeElimination, FoldConstant, RemoveUnusedFunctions
PO-0	AlterOpLayout, CombineParallelConv2D, DefuseOps, DynamicToStatic, CanonicalizeOps, CanonicalizeCast
PO-1	CanonicalizeCast, AlterOpLayout, DefuseOps, CombineParallelConv2D, PartitionGraph, FakeQuantizationToInteger
PO-2	ToMixedPrecision, CombineParallelConv2D, EliminateCommonSubexpr, SimplifyFCTranspose, CanonicalizeOps, DefuseOps, ToGraphNormalForm, ToGraphNormalForm
PO-3	CombineParallelDense, FakeQuantizationToInteger, AlterOpLayout, CombineParallelConv2D, ToGraphNormalForm, CanonicalizeOps

Experimental Set-Up

- Compile using passes available on TVM
  - TVM = Tensor Virtual Machine, a compiler stack for deep-learning models
- Evaluate on two GPUs (GeForce RTX 2080 and A100)

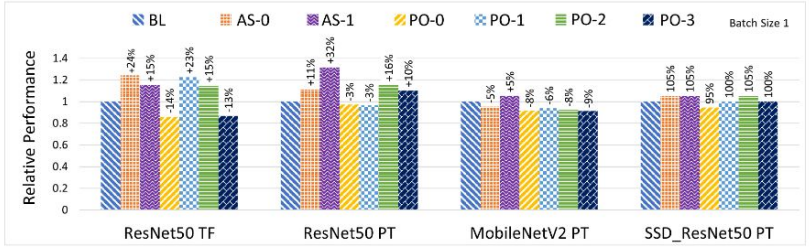
Results

- Metrics used for evaluation
  - Throughput
  - Latency
  - Compile Time
  - Power
  - Memory used
  - Temperature
- Evaluated the performance for the seven sets of passes
- Create computation graphs in TensorFlow, PyTorch, and MXNet

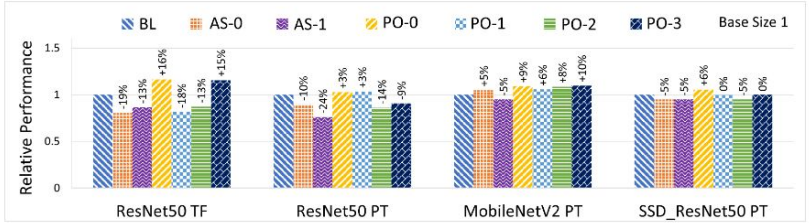
Results – Throughput and Latency

- For ResNet50 in TensorFlow and PyTorch, there was an improvement in throughput
- Throughput for MobileNet and SSD-ResNet did not see much improvement
  - MobileNet does not contain many relevant passes
- Neutral architecture-aware selection of optimization reduced the latency for ResNet50 in TensorFlow and Pytorch as well as for Mob ileNet and SSD_Resnet

5.1 Experiments on GeForce RTX 2080



(a) Variation of "Throughput" with Pass Selection



(b) Variation of "Latency" with Pass Selection

Results – Compile Time

- Pruning the optimization passes' search space ands selecting lesser, more relevant passes than optimization level 3 improves the compile time significantly
- Architecture-aware selection of passes was very effective for reducing compile-time
  - Nearly halved the compile time or Mobilenet and SSD_ResNet
  - BERT saw a 92% reduction in compile time
- Good implications for JIT compilation

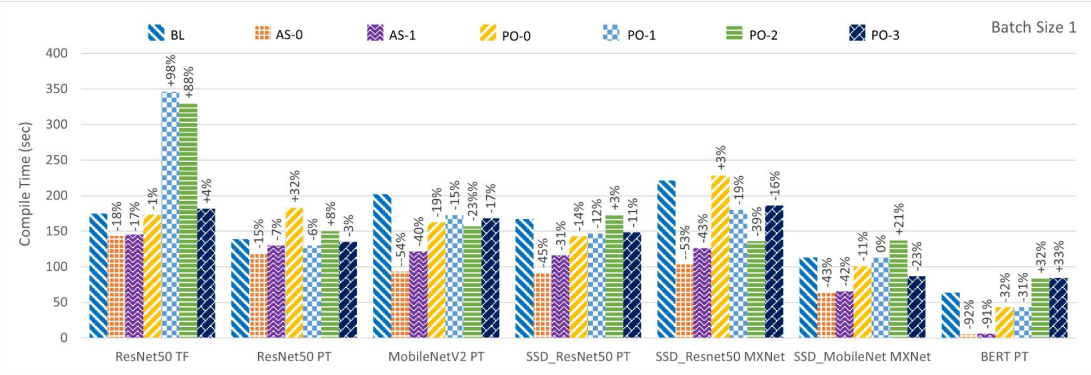


Figure 3: Compile-Time Reduction on A100 GPU

Tl;dr – Paper Contributions and Technique Novelty

- Generally improves throughput compared to the base optimization
- Technique of offline mapping passes to different neural networks significantly reduces the search space and therefore reduces the compile time
- Can target evaluation metrics for improvements
  - Ex: improve throughput vs. memory usage
- Improvements for memory utilization

Tl;dr – Paper Limitations

- Need to manually classify optimization passes to neural network architectures
  - Manual work grows as more optimization passes or neural network architectures are developed
- Authors experiment with only 4 different neural network architectures
- Authors are unclear on some areas on their paper
  - The process of identifying relevant classes for each pass
  - Some results of different software/computation graph generation combinations were likely omitted without a note on why
    - Some pass selections and their ordering are also omitted