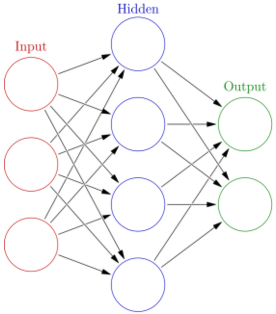
TVM: An Automated End-to-End Optimizing Compiler for Deep Learning

Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Meghan Cowan, Haichen Shen, Leyuan Wang, Yuwei Hu, Luis Ceze, Carlos Guestrin, Arvind Krishnamurthy

Presented by Aaron Solomon

Deep Learning - everywhere!

Old School:



CPU

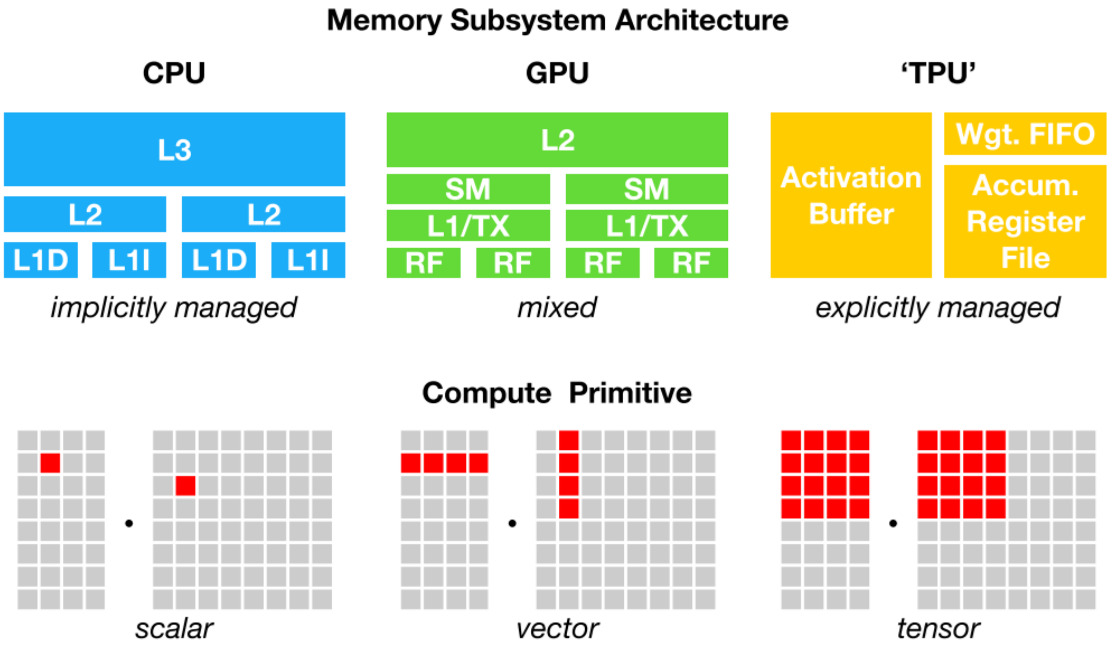
Today:

CPU

GPU

TPU

Fundamentally different memory architectures

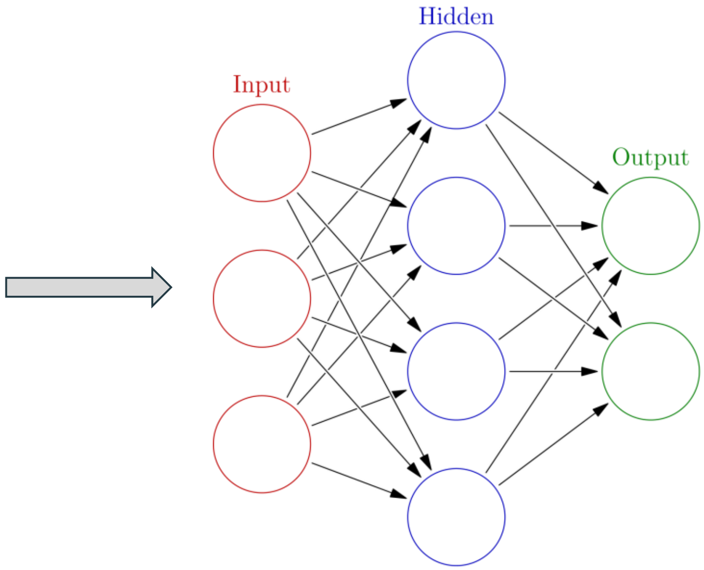


Challenges for Generalized Deep Learning

- Numerous hardware devices
  - GPUs, CPUs, TPUs, etc
- Bespoke low-level implementation needed to maximize efficiency on each ASIC/chip
- Many DL software solutions
  - Keras, TensorFlow, PyTorch, etc
- Lots of tuning
- Manual optimization is time intensive

Current Optimization

- Keras
- TensorFlow
- MXNet
- Caffe



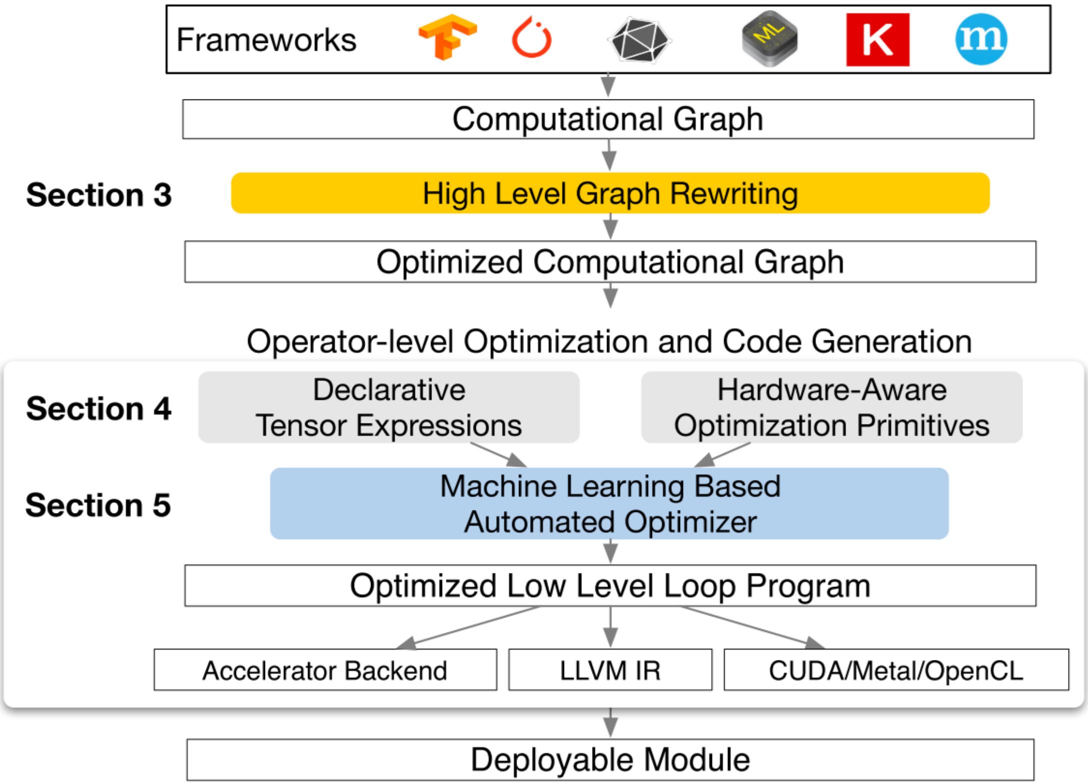
But graph optimization does not help low-level hardware efficiency!

Current architectures may perform high-level graph optimization and bespoke kernels

TVM

- Current SOA:
  - Each DL package implements bespoke code for kernels
  - High-level graph optim
- Goal: automate generation of optimized low-level code for many backends without human intervention by providing high-level (graph) and low-level optimizations
- Contributions
  - Graph Rewriter
  - Tensor Expression Language
  - Automated Program Optimization
  - Overall: automates time intensive process

TVM



Frameworks
Computational Graph
Section 3
High Level Graph Rewriting
Optimized Computational Graph
Operator-level Optimization and Code Generation
Section 4
Declarative Tensor Expressions
Hardware-Aware Optimization Primitives
Section 5
Machine Learning Based Automated Optimizer
Optimized Low Level Loop Program
Accelerator Backend
LLVM IR
CUDA/Metal/OpenCL
Deployable Module

Graph Level Modifications

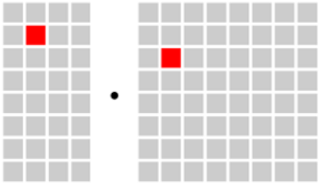
- Operator Fusion
  - Combines many small ops
- Constant Folding
  - Pre-computes static graphs
- Static Memory Planning Pass
  - Pre-allocates memory for needed tensors
- Data Layout Transformations
  - Optimize data storage for each backend

Operator Fusion

- Operator Types
  - One to one (addition)
  - Reduction (sum)
  - Complex-Out-Fusable (fuse element-wise)
  - Opaque (not-fusable)
- Specify rules for combining operators
- **Avoids intermediate memory storage**

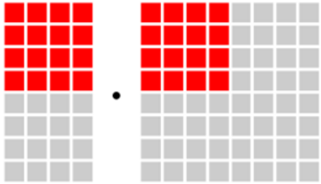
Data Layout Transforms

- Many possible storage options
  - What does the kernel use? 4 x 4 matrix or length 16 vector?
- Considers hardware-preferred data layout and optimizes if possible
- Transforms data between producer and consumer if unequivalent



scalar

CPU



tensor

TPU

Transforms if needed

Tensor Expression Language

- Specify products and operation, let TVM decide how to accomplish it

```
m, n, h = t.var('m'), t.var('n'), t.var('h')
A = t.placeholder((m, h), name='A')
B = t.placeholder((n, h), name='B')
k = t.reduce_axis((0, h), name='k')
C = t.compute((m, n), lambda y, x:
                   t.sum(A[k, y] * B[k, x], axis=k))
```

computing rule

result shape

- Many schedules proposed, inefficient ones culled

Nested Parallelism and Tensorization

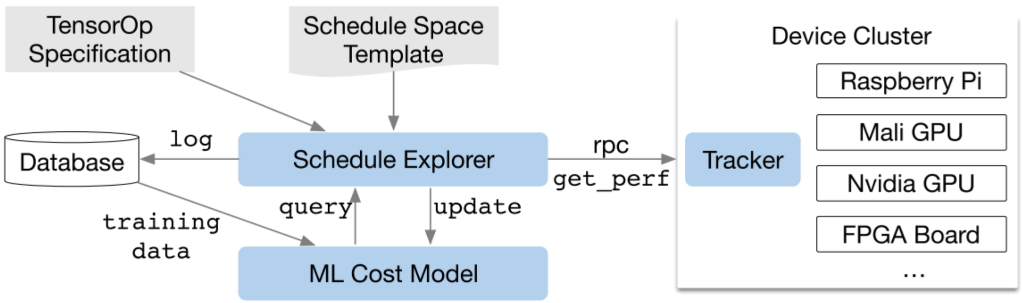
- Nested Parallelism
  - Explicit memory scopes enable multiple threads to share the same reference memory
  - Reduces fetch and mem transfer time
- Tensorization (compute primitives for tensors)
  - Uses specific language
  - Extensible - just specify hardware and the data representation it wants

Latency Hiding

- Simultaneous memory and compute ops to maximize efficiency
- CPUs
  - Multithreading
- GPUs
  - Context switching
- TPUs
  - Decoupled access/execute
- Virtual threading to control latency hiding

Automated Program Optimization

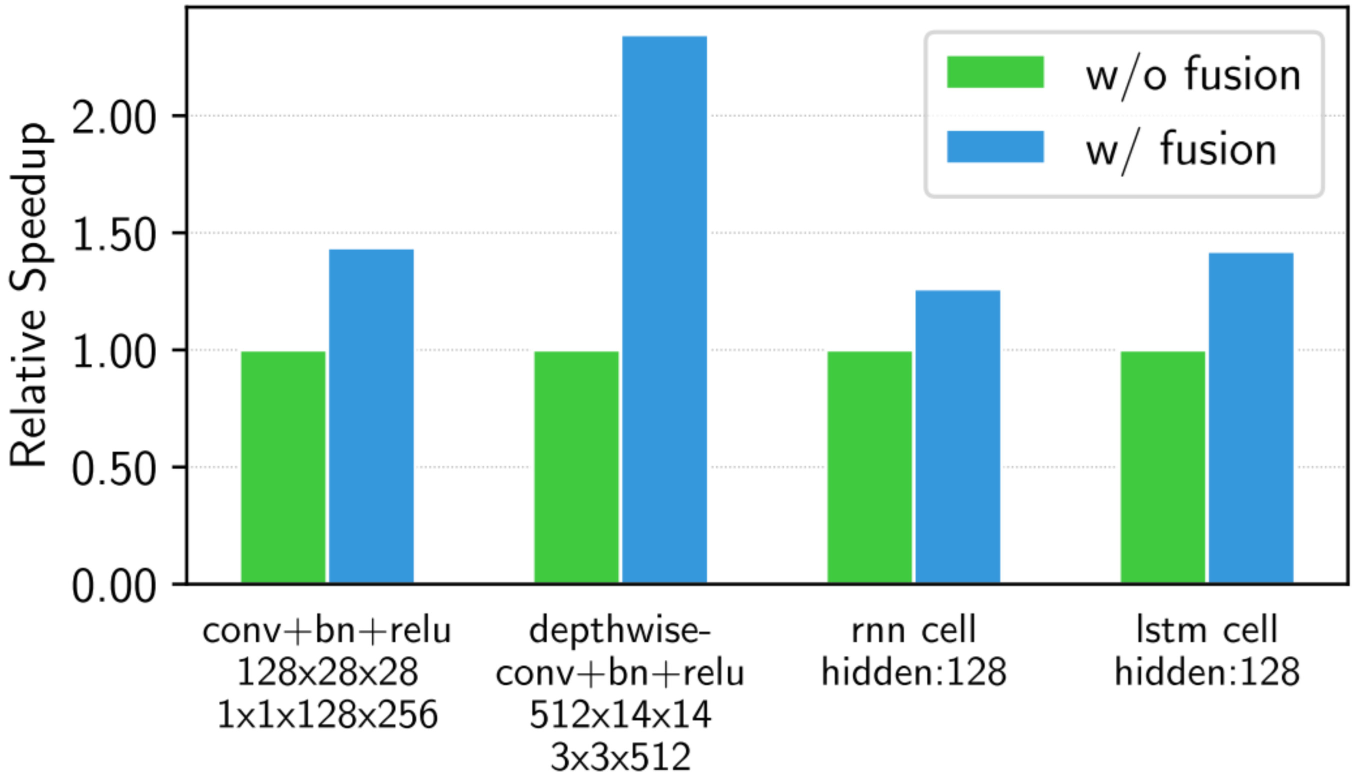
- So many pieces of code and scheduling primitives!
- Adversarial System
  - Part 1: Proposes new schedule configuration
  - Part 2: Predicts cost of proposed configuration



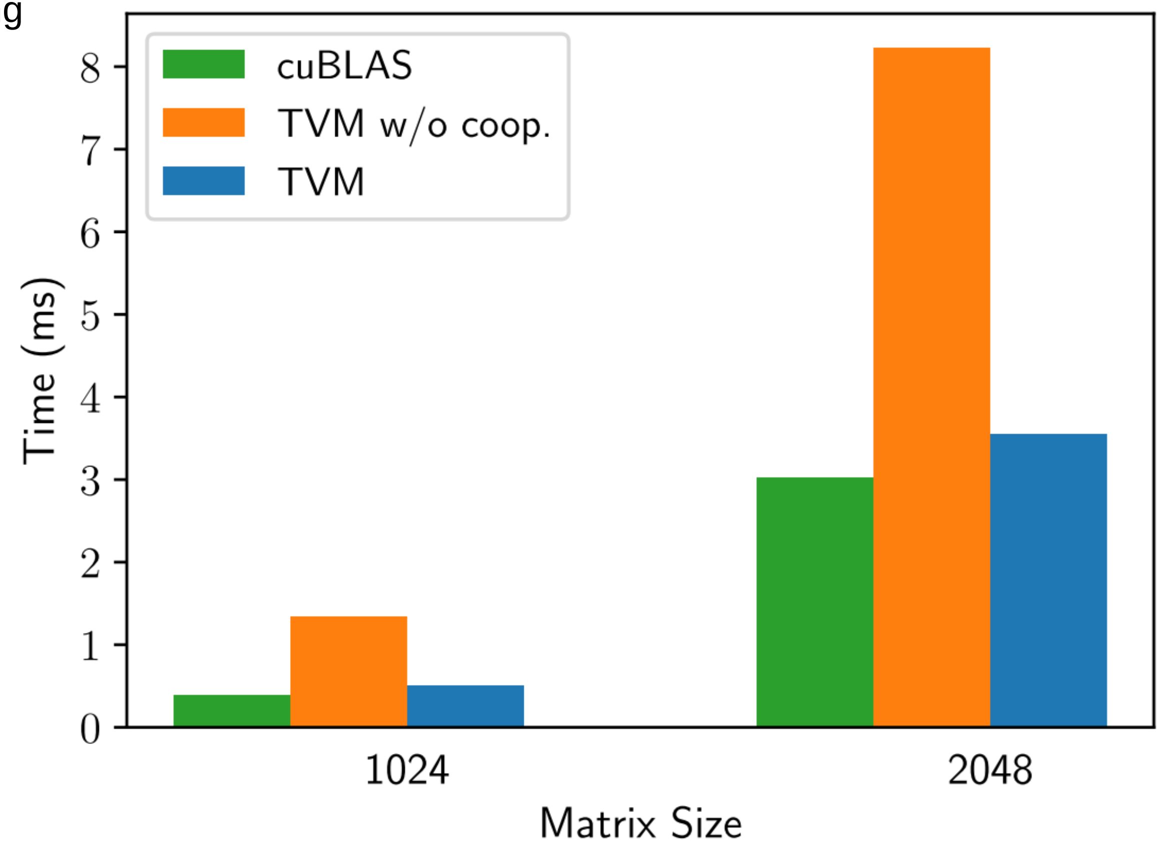
Automated Program Optimization

- Schedule Template Specification
  - Schedule = possible configuration
- One Hot Encoding of program features (loop elements, etc)
- Cost Model
- Simulated Annealing, Random Walks
- Gradient Tree Boosting
  - Input: Low Level Code
  - Output: Estimated (relative) time

Operator Fusion

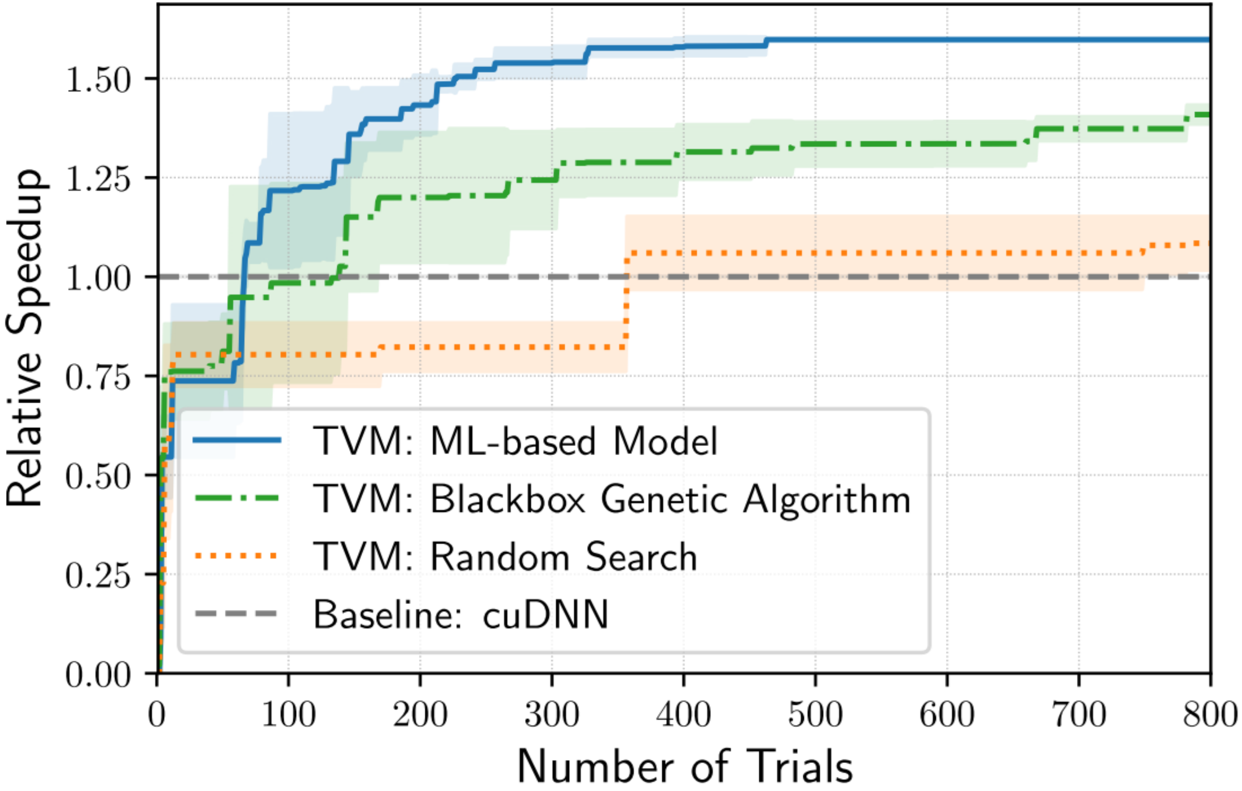


Mem Loading



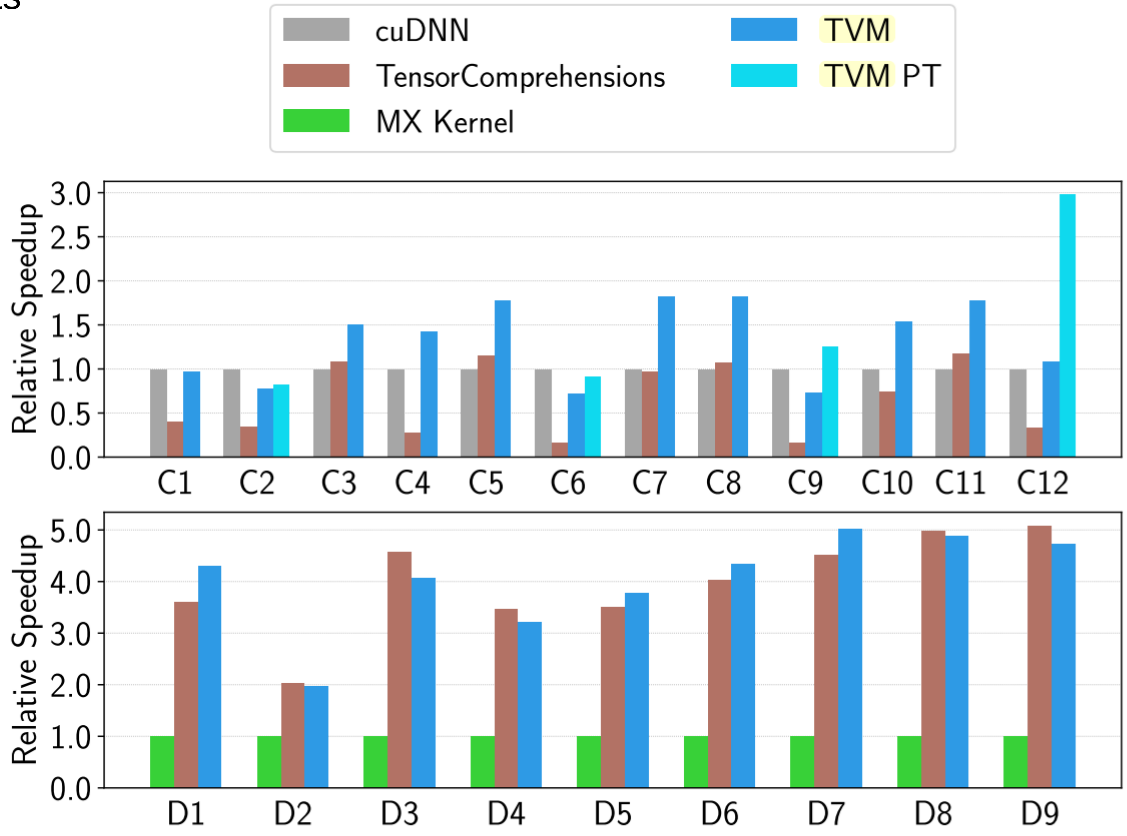
cuBLAS
TVM w/o coop.
TVM
Time (ms)
0
1
2
3
4
5
6
7
8
1024
2048
Matrix Size

Speed Up

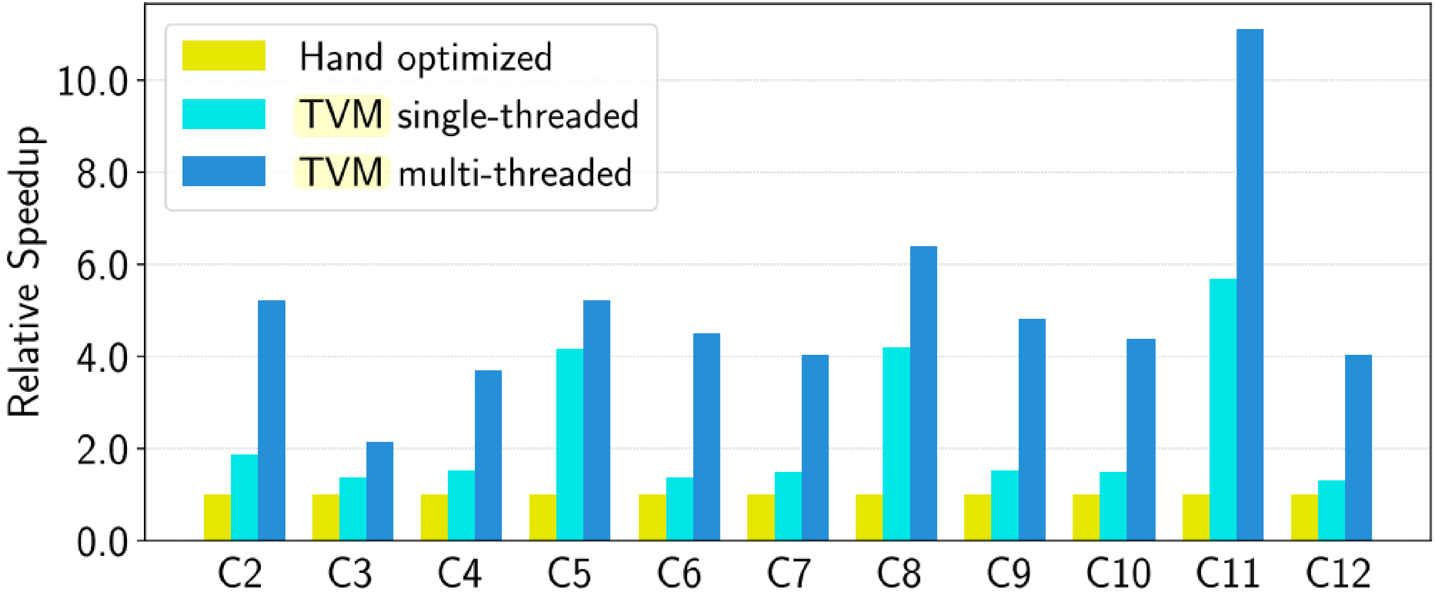


1.50
1.25
1.00
0.75
0.50
0.25
0.00
Relative Speedup
0
100
200
300
400
500
600
700
800
Number of Trials
TVM: ML-based Model
TVM: Blackbox Genetic Algorithm
TVM: Random Search
Baseline: cuDNN

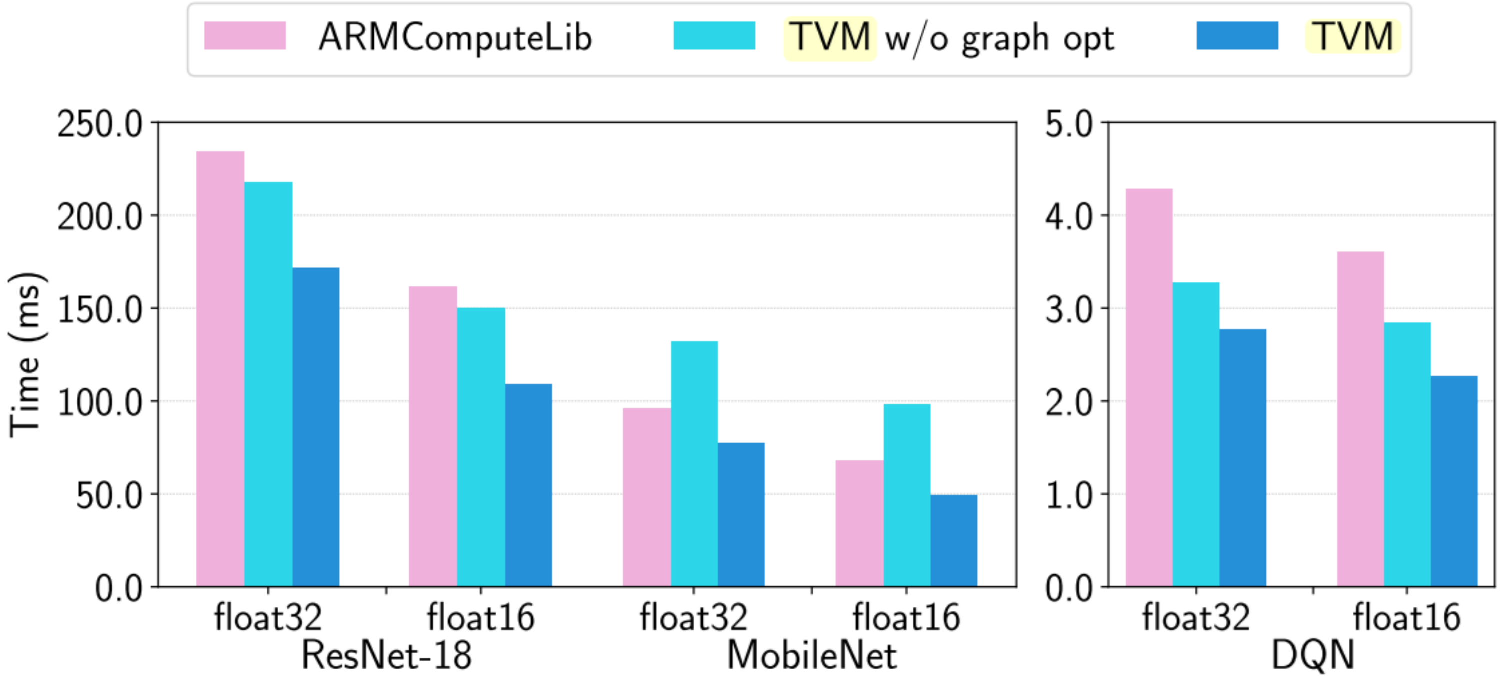
Conv Net Results



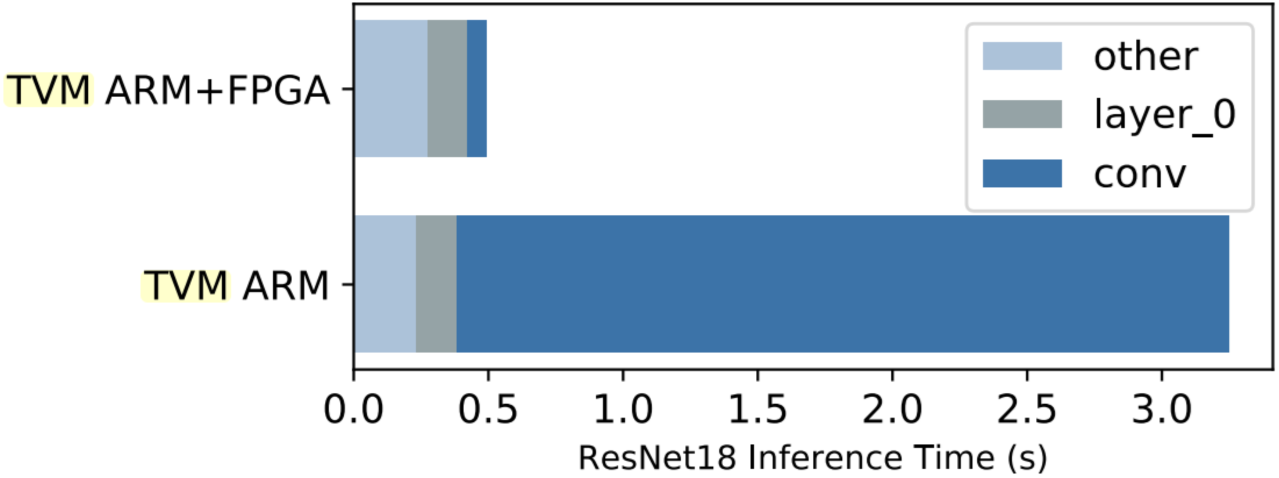
TVM MultiThread Capability



Mobile


ARMComputeLib
TVM w/o graph opt
TVM
Time (ms)
250.0
200.0
150.0
100.0
50.0
0.0
float32
float16
float32
float16
ResNet-18
MobileNet
5.0
4.0
3.0
2.0
1.0
0.0
float32
float16
DQN

VDLA/FPGA



TVM ARM+FPGA
TVM ARM
other
layer_0
conv
0.0
0.5
1.0
1.5
2.0
2.5
3.0
ResNet18 Inference Time (s)

Critique

- Good performance relative to baseline
- Not clear how much is actually novel
  - Other autotuners exist (ATLAS, FFTW, OpenTuner)
  - “Larger search space”
- Lack comparisons that actually demonstrate device generalizability that they seek
  - Should show TVM optimized systems vs. optimized package specific
- Automated work is sparse
  - Presented as “optimization with a side of automation” rather than an automation paper

Thank You!