Introduction to the TVM Open Source Deep Learning Compiler Stack

Luis Ceze

A perfect storm

Growing set of requirements: **Cost, latency, power, security & privacy**

Cambrian explosion of models, workloads, and use cases

<table>
<thead>
<tr>
<th>CNN</th>
<th>GAN</th>
<th>RNN</th>
<th>MLP</th>
<th>DQNN</th>
</tr>
</thead>
</table>

Rapidly evolving ML software ecosystem

Silicon scaling limitations (Dennard and Moore)

Cambrian explosion of HW backends. Heterogeneous HW
# Current Dominant Deep Learning Systems Landscape

## Orchestrators
- Kubeflow
- Seldon
- Algorithmia
- Azure ML
- GCP Datalab

## Frameworks and Inference engines
- TensorFlow
- PyTorch
- MXNet
- Keras
- ONNX
- TensorRT

## DL Compilers
- nGraph
- Glow
- XLA
- MLIR
- TVM

## Kernel Libraries
- cuDNN
- NNPack
- MKL-DNN
  - Hand optimized

## Hardware

---

© 2020 OctoML and University of Washington
End-to-end, framework to metal open stack. Research and deployment.

Open source synthesizable deep learning accelerator design
Automated by Machine Learning

High-Level Differentiable IR

Tensor Expression IR

LLVM, CUDA, Metal

VTA

Edge FPGA

Cloud FPGA

ASIC

ML-based Optimization

AutoTVM

AutoVTA

Hardware Fleet

TVM: Automated End-to-end Optimizations for Deep Learning. Chen et al. OSDI 18
import tvm
from tvm import relay

graph, params = Frontend.from_keras(keras_resnet50)

graph, lib, params = Relay.build(graph, target)

module = runtime.create(graph, lib, tvm.gpu(0))
module.set_input(**params)
module.run(data=data_array)
output = tvm.nd.empty(out_shape, ctx=tvm.gpu(0))
module.get_output(0, output)

input

Deployable Module

prediction tabby, tabby cat

End-user perspective: Compile & deploy
Open source: ~420+ contributors from UW, Berkeley, Cornell, UCLA, Amazon, Huawei, NTT, Facebook, Microsoft, Qualcomm, Alibaba, Intel, ...

Used in production at leading companies

- Deep Learning Compiler Service
- DSP/Tensor engine for mobile
- Mobile and Server Optimizations
- Cloud-side model optimization

Incubated as Apache TVM. Independent governance, allowing competitors to collaborate.
Existing Deep Learning Frameworks

Frameworks

- TensorFlow
- MXNet
- PyTorch
- Theano
- Keras
- MindSpore

Hardware

- NVIDIA

High-level data flow graph

Primitive Tensor operators such as Conv2D

Offload to heavily optimized DNN operator library

e.g. cuDNN
Engineering costs limits progress

New operator introduced by operator fusion optimization potential benefit: 1.5x speedup

cuDNN

Frameworks

Engineering intensive

© 2020 OctoML and University of Washington
Our approach: Learning-based Learning System

Frameworks

High-level data flow graph and optimizations

Machine Learning based Program Optimizer

Hardware

Directly generate optimized program for new operator workloads and hardware
Tensor Compilation/Optimization as a search problem

Tensor Expression (Specification)

\[
C = \text{tvm.compute}((m, n), \\
\quad \text{lambda } y, x: \text{tvm.sum}(A[k, y] \ast B[k, x], \text{axis} = k))
\]

Search Space of Possible Program Optimizations

Low-level Program Variants

```
inp_buffer AL[8][8], BL[8][8]
acc_buffer CL[8][8]
for yo in range(128):
    for xo in range(128):
        for ko in range(128):
            vda.dma_copy2d(AL, A[ko+8:ko+8][yo+8:yo+8])
            vda.dma_copy2d(BL, B[ko+8:ko+8][xo+8:xo+8])
            vda.fused_gemm8x8_add(CL, AL, BL)
            vda.dma_copy2d(CL[yo+8:yo+8][xo+8:xo+8], CL)
```

```
for yo in range(128):
    for xo in range(128):
        for ko in range(128):
            for yi in range(8):
                for xi in range(8):
                    vda.fused_gemm8x8_add(CL[yo+8:yo+8][xo+8:xo+8], A[ko+8+ki], B[ko+8+ki][yo+8+yi])
```

```
for y in range(1024):
    for x in range(1024):
        C[y][x] = 0
        for k in range(1024):
            C[y][x] += A[k][y] \ast B[k][x]
```

Search Space Example (1/3)

**Tensor Expression (Specification)**

\[
C = \text{tvm.compute}((m, n), \\
\lambda y, x: \text{tvm.sum}(A[k, y] \ast B[k, x], \text{axis}=k))
\]

**Search Space of Possible Program Optimizations**

**Vanilla Code**

```python
for y in range(1024):
    for x in range(1024):
        C[y][x] = 0
    for k in range(1024):
        C[y][x] += A[k][y] \ast B[k][x]
```
Search Space Example (2/3)

Tensor Expression (Specification)

\[
C = \text{tvm.compute}((m, n), \lambda y, x: \text{tvm.sum}(A[k, y] \times B[k, x], \text{axis}=k))
\]

Search Space of Possible Program Optimizations

Loop Tiling for Locality

```python
for yo in range(128):
    for xo in range(128):
        C[yo*8:yo*8+8][xo*8:xo*8+8] = 0
    for yi in range(8):
        for xi in range(8):
            for ki in range(8):
                C[yo*8+yi][xo*8+xi] += A[ko*8+ki][yo*8+yi] \times B[ko*8+ki][xo*8+xi]
```
Search Space Example (3/3)

Tensor Expression (Specification)

\[
C = \text{tvm.compute}((m, n), \\
\text{lambda } y, x: \text{tvm.sum}(A[k, y] * B[k, x], \text{axis}=k))
\]

Map to Accelerators

```python
inp_buffer AL[8][8], BL[8][8]
acc_buffer CL[8][8]
for yo in range(128):
    for xo in range(128):
        vsla.fill_zero(CL)
        for ko in range(128):
            vsla.dma_copy2d(AL, A[ko*8:ko*8+8][yo*8:yo*8+8])
            vsla.dma_copy2d(BL, B[ko*8:ko*8+8][xo*8:xo*8+8])
            vsla.fused_gemm8x8_add(CL, AL, BL)
        vsla.dma_copy2d(C[yo*8:yo*8+8,xo*8:xo*8+8], CL)
```
Optimization space is really large…

Tensor Expression (Specification)

\[
C = \text{tvm.compute}((m, n), \\
\text{lambda } y, x: \text{tvm.sum}(A[k, y] * B[k, x], \text{axis}=k))
\]

Billions of possible optimization choices

Typically explored via human intuition. How can we automate this? Auto-tuning is too slow.
Problem Formalization

Expression $e$ to $S_e$ is mapped to AutoOpt, which maps to $C$. Then $C$ is mapped to Code Generator, which generates the program $x$. The objective is to minimize $f(x)$ with respect to $c \in S_e$.

$$x = g(e, c)$$

$$\arg\min_{c \in S_e} f(g(e, c))$$

Cost: Execute Time
Black-box Optimization

Try each configuration $C$ until we find a good one

Challenge: Lots of experimental trials, each trial costs ~1 second
Cost-model Driven Approach

Use cost model to pick configuration

$e$  Expression  $c$  AutoOpt  $C$  Code Generator  $x$

$S_e$  Search Space  $C$  Cost Model  $\hat{f}(e, c)$

Challenge: Need reliable cost model per hardware
Our approach: Use machine learning to learn a statistical cost model

**Benefit:** Automatically adapt to hardware type

**Important:** How to design the cost model
AutoTVM Overview

**Conv2D**

- **Expression:** $e$
- **Search Space:** $S_e$
- **AutoTVM**
- **Statistical Cost Model**
- **Code Generator**
- **Learning**
- **Training data**
- **$f(x)$**

**Matmul**

- **Expression:** $e_2$
- **Search Space:** $S_{e_2}$
- **Shared Cost Model**
- **AutoTVM**
- **Code Generator**

**Benefit:** Low-level AST is a common representation (General, task invariant)

<table>
<thead>
<tr>
<th>Task Invariant</th>
<th>Time Cost</th>
<th>Predictive Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vanilla Model</td>
<td>No</td>
<td>Low</td>
</tr>
<tr>
<td>Tree-based Model</td>
<td>Yes</td>
<td>Low</td>
</tr>
<tr>
<td>Neural Model</td>
<td>Yes</td>
<td>High</td>
</tr>
</tbody>
</table>

O(microseconds) inference vs. O(seconds) execution

Learning to Optimize Tensor Programs. Chen et al. NeurIPS 18
Does it work?

Better than hand-tuned code in a few minutes

1.50x faster than hand-tuned in steady state

3x to 10x faster tuning w/ transfer learning
Device Fleet: Distributed Test Bed for AutoTVM

Resource Manager (Tracker)

AutoTVM Experiment 1

AutoTVM Experiment 2
State-of-the-art performance

Backed by cuDNN

- **TensorFlow**
- **Apache MXNet**
- **TVM**
- **TensorFlow-XLA**

**Nvidia Titan X**

- Competitive on standard models
- 3x better on emerging models

**ARM CPU (Cortex-A53)**

**ARM GPU (Mali)**

Special frameworks for the particular hardware platform

- **TensorFlow Lite**
- **ArmComputeLib**
- **TVM**

Key point: TVM offers good performance with low manual effort
End-to-end, framework to metal open stack. Research and deployment

High-Level Differentiable IR

Tensor Expression IR

LLVM, CUDA, Metal

VTA

Edge FPGA
Cloud FPGA
ASIC

Open source synthesizable deep learning accelerator design
DL Accelerator Design Challenges

- Keeping up with algorithmic changes
  - (VTA: two-level ISA, templatized design)

- Finding the right generality/efficiency trade-off
  - (VTA: templatized design + HW parameter search)

- Enable a “day-0” software stack on top
  - (VTA: tight coupling with TVM)
## VTA: Open & Flexible Deep Learning Accelerator

### Current TVM Stack

- **VTA Runtime & JIT Compiler**
- **VTA Hardware/Software Interface (ISA)**
- **VTA MicroArchitecture**
- **VTA Simulator**

### Key Features

- Move hardware complexity to software via a **two-level ISA**
- Runtime **JIT-compile accelerator micro code**
- Native support in TVM
- Support heterogenous devices (split graph)
- Support for secure execution (soon)
• Decoupled access-execute with explicit software control
• Two-level ISA: JIT breaks multi-cycle “CISC” instructions into micro-ops
  • Enables model retargeting without HW changes
• Focused on FPGA deployments so far. Exploring custom silicon possibilities

µTVM - Bare-metal model deployment for edge devices

Optimize, compile and package model for standalone bare metal deployment

ML model → µTVM → Optimized model → Optimized operators → Standalone runtime → Flash code → Edge device board (ARM, MIPS, RISC-V,...)

See recent demo on TVM for Azure Sphere deployment.
Coming Soon - Ultra low bit-width quantization

Automatic quantization: 5-20x performance gains with reasonable accuracy loss.

TVM supports flexible code generation for a variety of data types
What about training?

- Direct support for training in Apache TVM coming soon!
- Automatic generation of gradient programs
- Support for customized data types and training on FPGAs
Other Ongoing TVM efforts

• Autoscheduling (Zheng et al. OSDI’20 @ UCBerkeley)
• Automatic synthesis of operator implementations (Cowan et al. CGO’20 @ UWash)
• Sparse support (NLP, graph convolutional neural networks, etc...)
• Secure enclaves
• ...
• Join the community!
2nd TVM conference on Dec 5, 2019. 200+ ppl last year!

• Video tutorials
• iPython notebooks tutorials

Drive TVM adoption
Core infrastructure and improvements

Product: SaaS automation for ML ops
Optimizing, benchmarking, and packaging models for deployment

Support
TVM end users and hardware vendors

Apache TVM ecosystem
OctoML

https://octoml.ai

© 2020 OctoML and University of Washington
What I would like you to remember...

TVM is an emerging **open source** standard for ML compilation and optimization

TVM offers

- Improved time to market for ML
- Performance
- Unified support for CPU, GPU, Accelerators
- On the framework of your choice

OctoML is here to help you succeed in your ML deployment needs