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


Presented by Aaron Solomon
Deep Learning - everywhere!

Old School:

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
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-Fusible (fuse element-wise)
  - Opaque (not-fusible)
- 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

CPU

Transforms if needed

TPU

**scalar**

**tensor**
Tensor Expression Language

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

```python
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))
```

- 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

![Bar chart showing relative speedup for different operations with and without fusion.](chart)

- **conv+bn+relu 128x28x28 1x1x128x256**
- **depthwise-conv+bn+relu 512x14x14 3x3x512**
- **rnn cell hidden:128**
- **Istm cell hidden:128**
Conv Net Results

[Chart showing relative speedup for different convolutional neural network results using cuDNN, TVM, TVM PT, and MX Kernel, comparing C1 to C12 and D1 to D9.]
TVM MultiThread Capability

- Hand optimized
- TVM single-threaded
- TVM multi-threaded

Relative Speedup vs. Components (C2 to C12)
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!