Hello!

Tensor

Virtual Machine

↳ LLVM

CS 744: TVM

Shivaram Venkataraman

Fall 2020

ADMINISTRIVIA

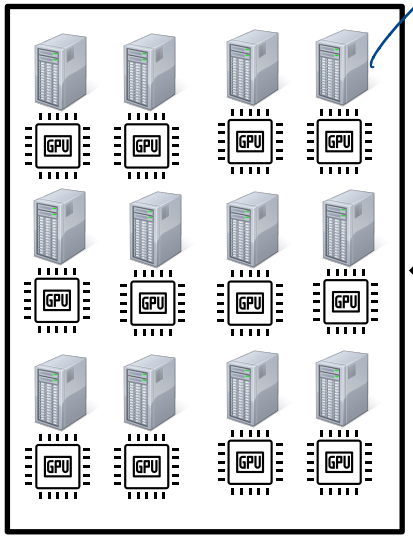
~~Assignments are done~~

- Course project titles
- Project proposal aka Introduction (10/16)
  - Introduction
  - Related Work
  - Timeline (with eval plan)

→ 2 page writeup

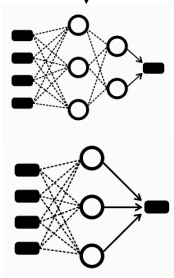
- Midterm: Oct 22

MACHINE LEARNING: STACK



Distributed

Train



make distributed training easy and scalable

movie recommend

Translations

Inference just the forward pass

Interplay inference & training

Scaling inference

Hardware

MOTIVATION: PERFORAMNCE PORTABILITY

Different hardware has different mem hierarchy/ compute primitives

You want high performance across hardware backends

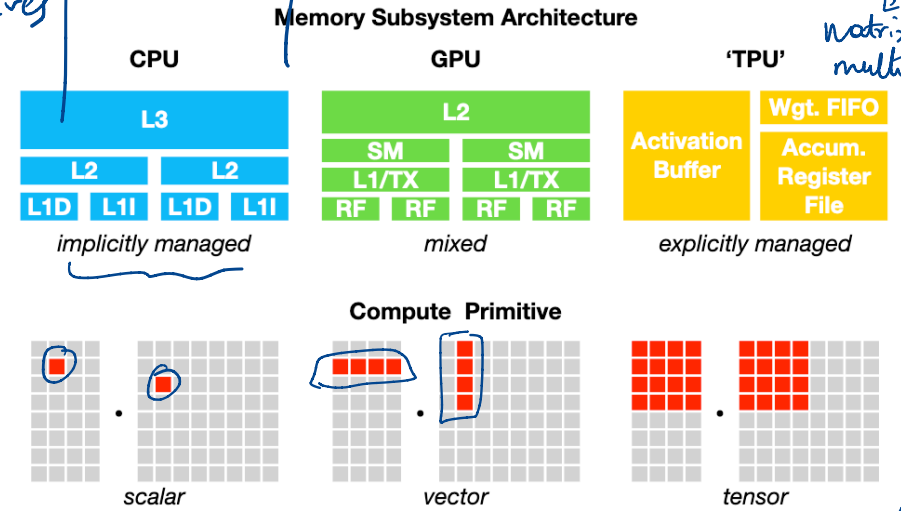
Dependence on vendor specific libraries

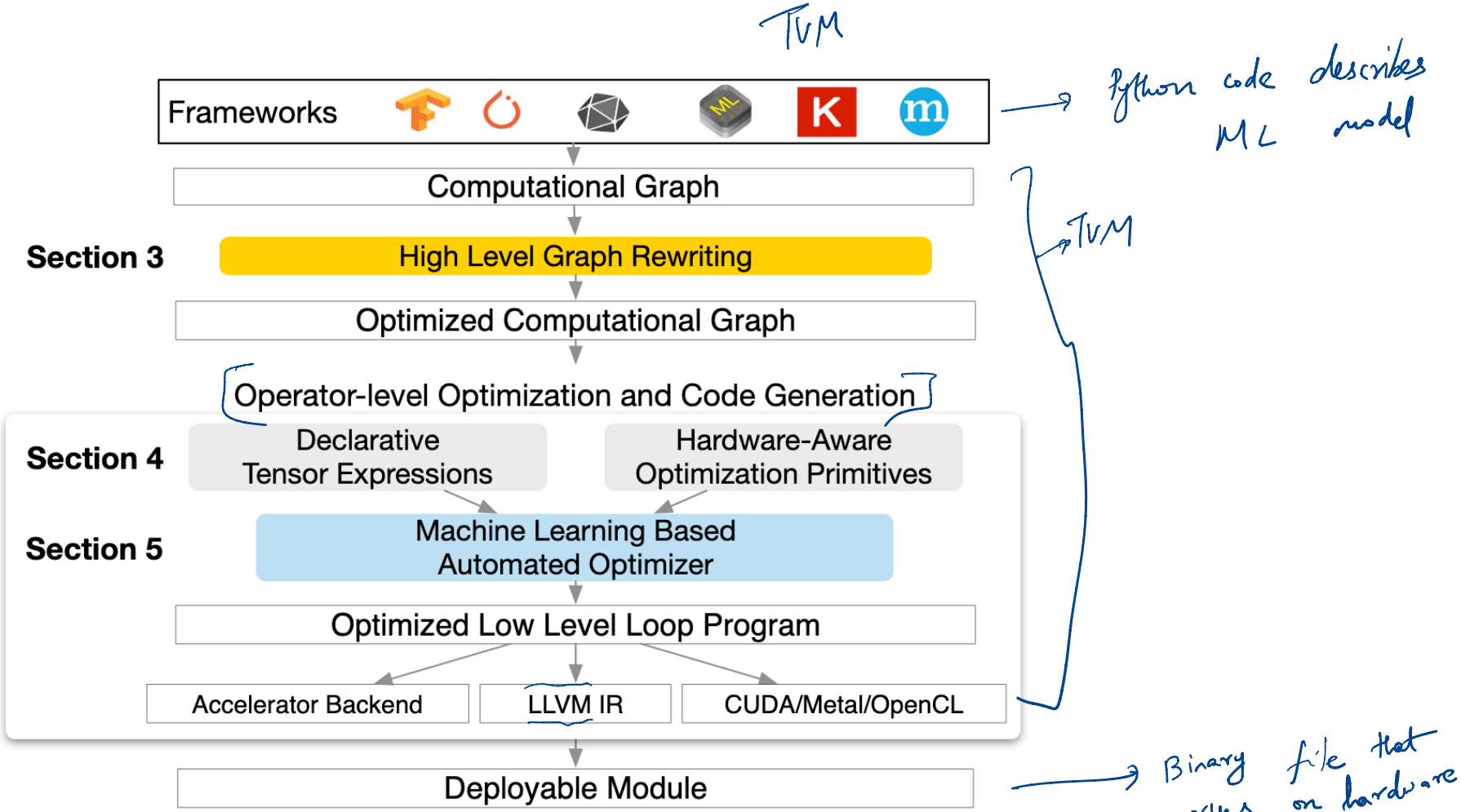
ML models evolve fast ⇒ new operators / new combination of operators } ⇒ not available in existing vendor libraries

Pytorch → model file

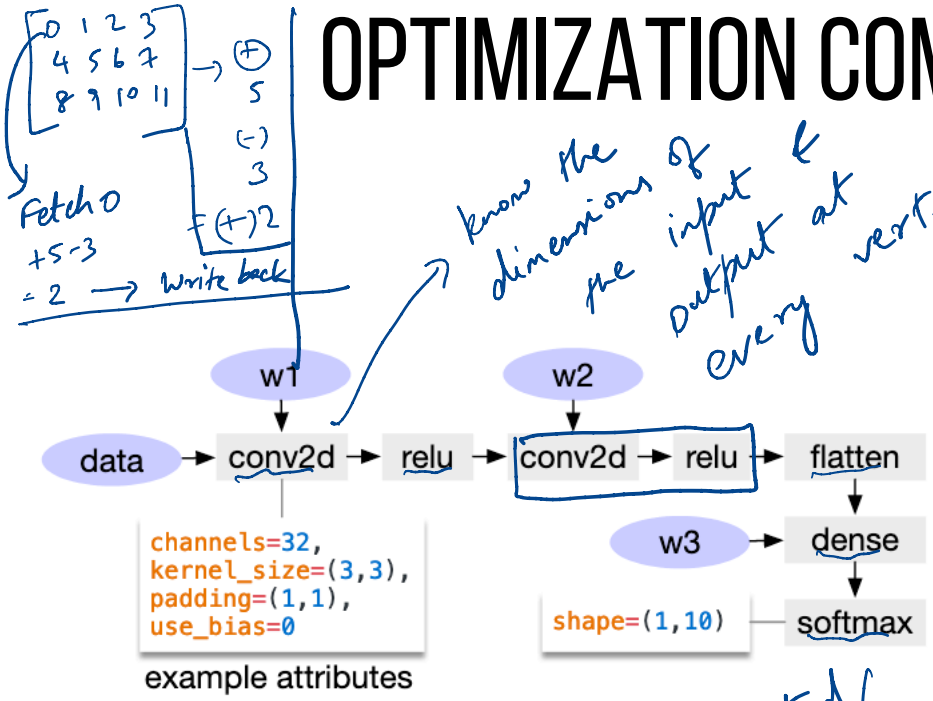
CPU ← Intel MKL

MKL: matrix multiply, conv 2d




TVM
Frameworks
Python code describes ML model
Computational Graph
TVM
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
Binary file that runs on hardware

OPTIMIZATION COMPUTATION GRAPHS



know the dimensions of the input & output at every vertex

2-layer NN that is represented as a graph

(1) Operator Fusion

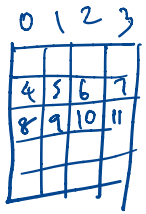
↳ Combine any graph vertexes into one low level op

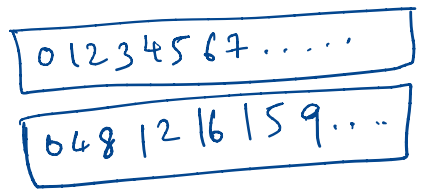
→ 1-1 operators, "map" → Combine ↳ (Spark map)

→ Sum reduction, scaling after

Data layout

Row Major, Column Major, Blocked,





Insert op change data layout

TENSOR EXPRESSION LANGUAGE

operators () expressed in tensor expression language

```
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))
```

result shape

computing rule

tensor

math operations

Common Arithmetic, Math operations

Know the shape of the output and the data accessed

CODE GENERATION

Halide → expression → schedule of instructions

OpenMP ← gcc

```
for thread_group (by, bx) in cross(64, 64):
  for thread_item (ty, tx) in cross(2, 2):
    local CL[8][8] = 0
    shared AS[2][8], BS[2][8]
    for k in range(1024):
      for i in range(4):
        AS[ty][i*4+tx] = A[k][by*64+ty*8+i*4+tx]
      for each i in 0..4:
        BS[ty][i*4+tx] = B[k][bx*64+ty*8+i*4+tx]
      memory_barrier_among_threads()
```

GPU "cache"

Co-operative

threads can use AS, BS to do computation

launch 50 threads
each thread does
load a[i,j], b[i,j]
compute
write it back

Nested parallelism → Cooperative parallelism

```
for i in 1:10
  for j in 1:5
    a[i,j] = b[i,j]+2
```

→ independent of other loop iterations

```
def gemm_intrin_lower(inputs, outputs):
    ww_ptr = inputs[0].access_ptr("r")
    xx_ptr = inputs[1].access_ptr("r")
    zz_ptr = outputs[0].access_ptr("w")
    compute = t.hardware_intrin("gemm8x8", ww_ptr, xx_ptr, zz_ptr)
    reset = t.hardware_intrin("fill_zero", zz_ptr)
    update = t.hardware_intrin("fuse_gemm8x8_add", ww_ptr, xx_ptr, zz_ptr)
    return compute, reset, update

gemm8x8 = t.decl_tensor_intrin(y.op, gemm_intrin_lower)
```

load, store, add

Tensorization → What is the hardware instruction set

Extensible!

Allows you to register operator intrinsics

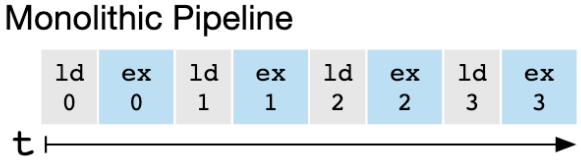
LATENCY HIDING

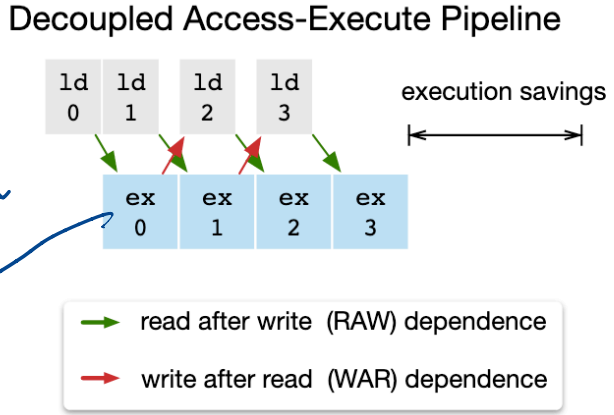
Same as Pytorch etc.

What is the goal?

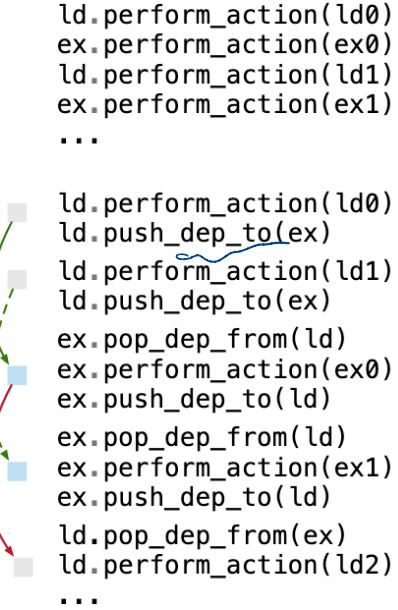
↳ Overlap computation and communication

Schedule that utilizes memory bandwidth & compute units





this execution happens after this load



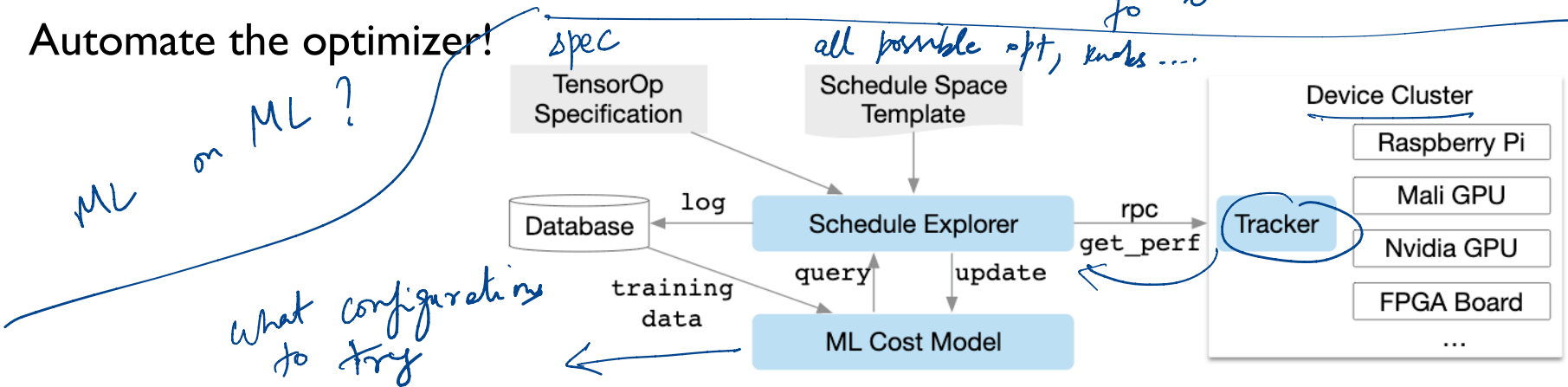
AUTOMATING OPTIMIZATION

Goal: Create a specialized operator for input shape and layout

Challenge:

- Choose appropriate schedule optimizations
- Tiling size, loop unrolling . . . . . lots of different choices & also lots of parameters / knobs to choose.

Automate the optimizer!



ML-BASED COST MODEL

Machine Learning Model Design Choices

- Speed: Faster than time it takes to evaluate a config (ms to ~ seconds)
- Quality: Use a rank objective to predict the relative order of runtime

Gradient tree boosting model (take code, generate features)

- memory access count
- reuse ratio of each memory buffer at each loop level
- one-hot encoding of loop annotations

ML-BASED COST MODEL

model perf when using config

<C1, 10ms>
<C2, 20ms>

Iteration

(1) Select a batch of candidates → each candidate is a configuration / schedule <C3, 8ms>
<C4, fail>

(2) Collect data ← Run on real hardware

(3) Use as training data to update the model

How to select candidates? → Step (1) above

Parallel Simulated Annealing

Start from a random config

Walk to a nearby config → $C_3 \rightarrow C_3'$ ask model is C_3' better than C_3

Yes → go & try C_3' on cluster

No → generate another config

Successful if cost decreases Else Reject

training data to GBT

DISTRIBUTED DEVICE POOL

Pool of devices to speed up profiling

RPC interface to run a trial on device

Share device pools for multiple graphs

SUMMARY

TVM: Compiler for ML inference models

Support high performance for range of models, hardware devices

Key ideas

- Graph-level optimizations → operator fusion
- Tensor expression language: Code-gen, Latency hiding etc
  ML based Cost Model for automation

DISCUSSION

https://forms.gle/WiVgJ3abGXXgfBN99

Consider that you are building an optimizer for Spark programs instead of ML inference. What would be some configuration knobs that you could similarly tune? What might be different from the TVM optimizer?

Similar logic → latency hiding
overlap comp, communication

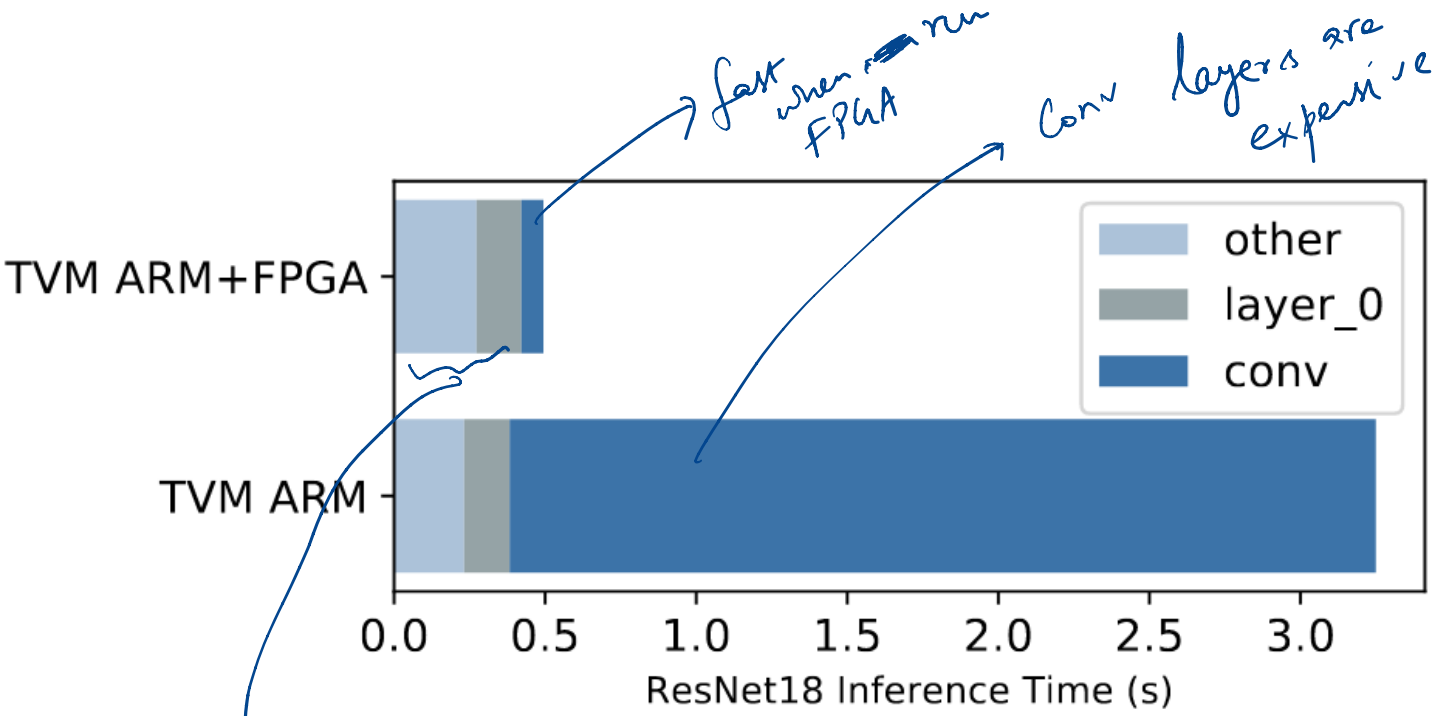
In TVM
↳ dimensions, access patterns

operator fusion → map & operations
↳ lazy!
↳ operators are user defined
challenging !?

Partitioning → Can you automate
↳ number of partitions / co-partitioning
↳ Performance!

Persistence → manually insert rdd.cache() → config space!

What is your takeaway from the following figure?



NEXT STEPS

Next class: Ray

Course project: Oct 16 (introductions)

Midterm: Oct 22