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

R244: Large-Scale Data Processing and Optimisation

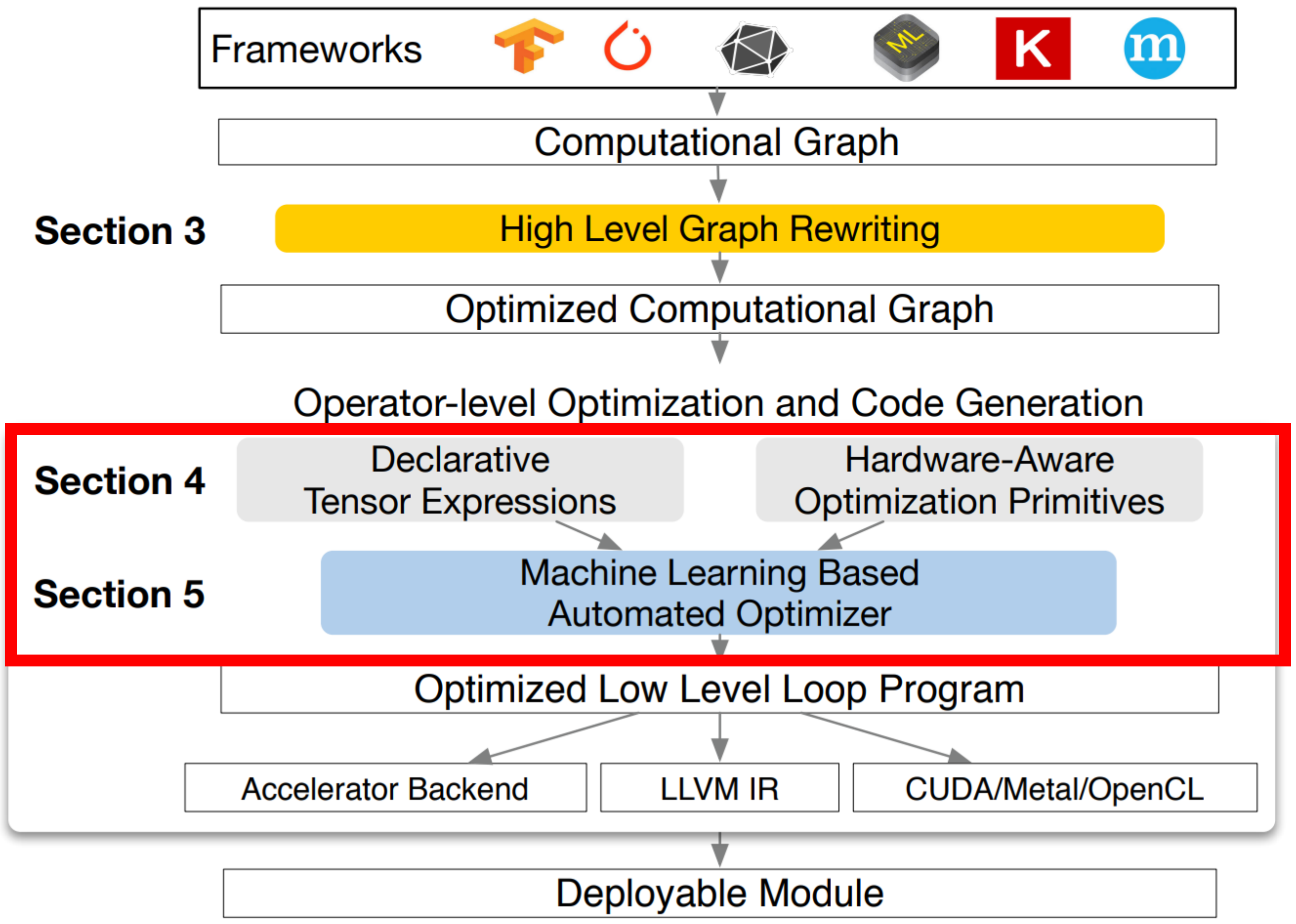
Kian Cross

Chen, T., Moreau, T., Jiang, Z., Zheng, L., Yan, E., Shen, H., ... & Krishnamurthy, A. (2018). {TVM}: An automated {End-to-End} optimizing compiler for deep learning. In 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18) (pp. 578-594).

Background

- There is an increasing need to bring machine learning to a wide diversity of hardware devices.
- This currently requires lots of manual effort to port to different backends (e.g. CPUs, GPUs, FPGAs).
- Previous work required manual effort and hand-optimisation (e.g., TensorFlow Lite).
- TVM is proposed to automate this process.

System Overview



Tensor Expression Language

- The language does not specify the loop structure and many other execution details.
- Provides flexibility for adding hardware-aware optimisations for various backends.

```
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))
```

Scheduling

- A schedule maps the tensor expressions to low-level code.
- Schedule transformations can be used to apply optimisations.
- There are many equivalent schedules.

```
A = t.placeholder((1024, 1024))
B = t.placeholder((1024, 1024))
k = t.reduce_axis((0, 1024))
C = t.compute((1024, 1024), lambda y, x:
              t.sum(A[k, y] * B[k, x], axis=k))
s = t.create_schedule(C.op)
```

```
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] * B[k][x]
```

+ Loop Tiling

```
yo, xo, ko, yi, xi, ki = s[C].tile(y, x, k, 8, 8, 8)
```

```
for yo in range(128):
  for xo in range(128):
    C[yo*8:yo*8+8][xo*8:xo*8+8] = 0
    for ko in range(128):
      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] * B[ko*8+ki][xo*8+xi]
```

+ Cache Data on Accelerator Special Buffer

```
CL = s.cache_write(C, vdla.acc_buffer)
AL = s.cache_read(A, vdla.inp_buffer)
# additional schedule steps omitted …
```

+ Map to Accelerator Tensor Instructions

```
s[CL].tensorize(yi, vdla.gemm8x8)
```

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

schedule — schedule transformation — corresponding low-level code

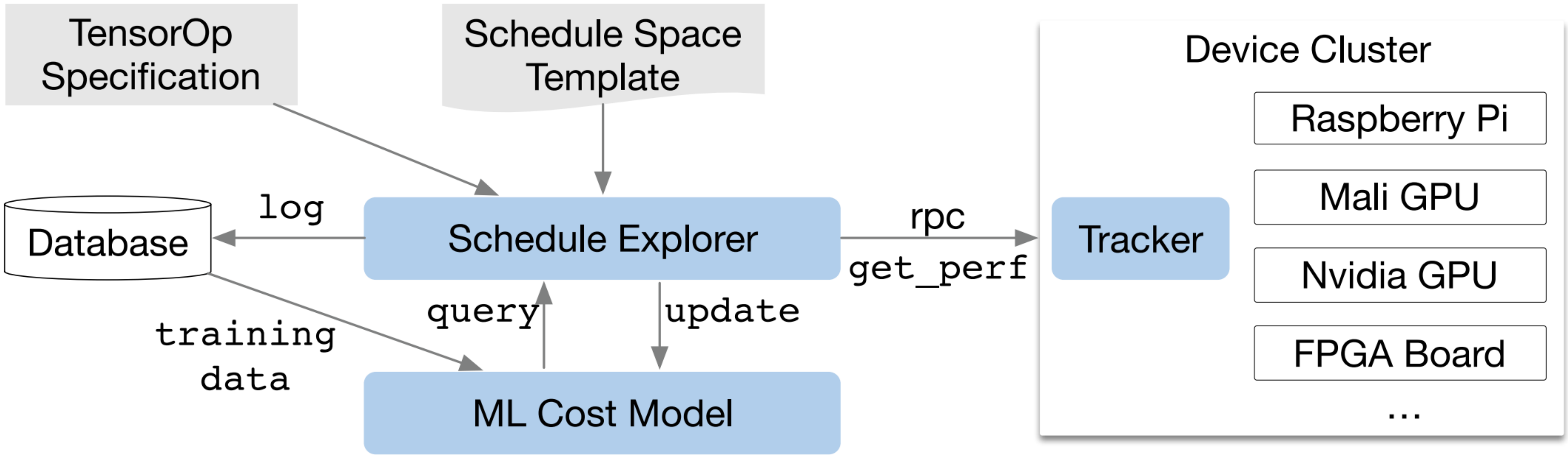
Schedule Transformations (Optimisations)

- Nested Parallelism with Cooperation
  - Groups of threads can cooperatively fetch the data they all need and place it into a shared memory space.
  - Takes advantage of the GPU memory hierarchy.
- Tensorization
  - Utilise tensor compute primitives.
- Explicit Memory Latency Hiding
  - Overlap memory operations with computation to maximize utilization of memory and compute resources.

How does TVM select the correct schedule?

Automated Optimisation

- Schedule Space Specification
  - Allows developer to incorporate domain specific knowledge to restrict the search space.
  - Each hardware backend is specified by a 'master template'.
- ML-Based Cost Model
  - Schedule explorer proposes configurations that may improve an operator's performance.
  - ML model takes this programme and predicts its running time.
  - Model is trained using runtime measurement data collected during exploration.
- Schedule Exploration
  - Promising candidates are run on hardware to obtain real measurements for training.


TensorOp
Specification
Schedule Space
Template
Device Cluster
Raspberry Pi
log
Database
Schedule Explorer
rpc
get_perf
Tracker
Mali GPU
Nvidia GPU
query
update
training
data
ML Cost Model
FPGA Board
...

By the end of all this, you have low-level hardware optimised code.

Evaluation

- "Experimental results show that TVM delivers performance across hardware back-ends that are competitive with state-of-the-art, hand-tuned libraries for low-power CPU, mobile GPU, and server-class GPUs."

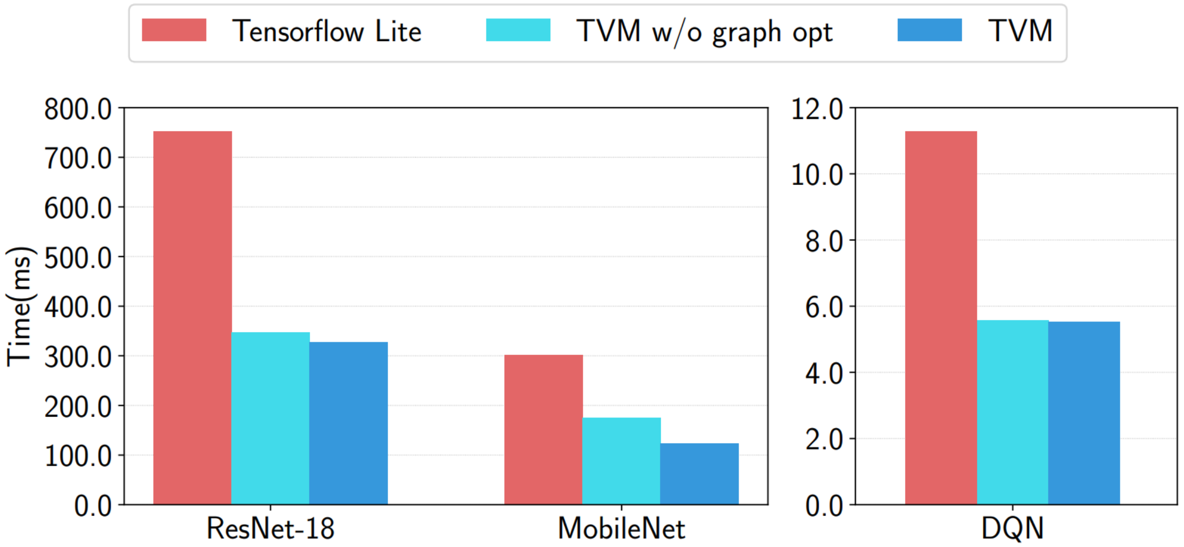


Figure 16: ARM A53 end-to-end evaluation of TVM and TFLite.

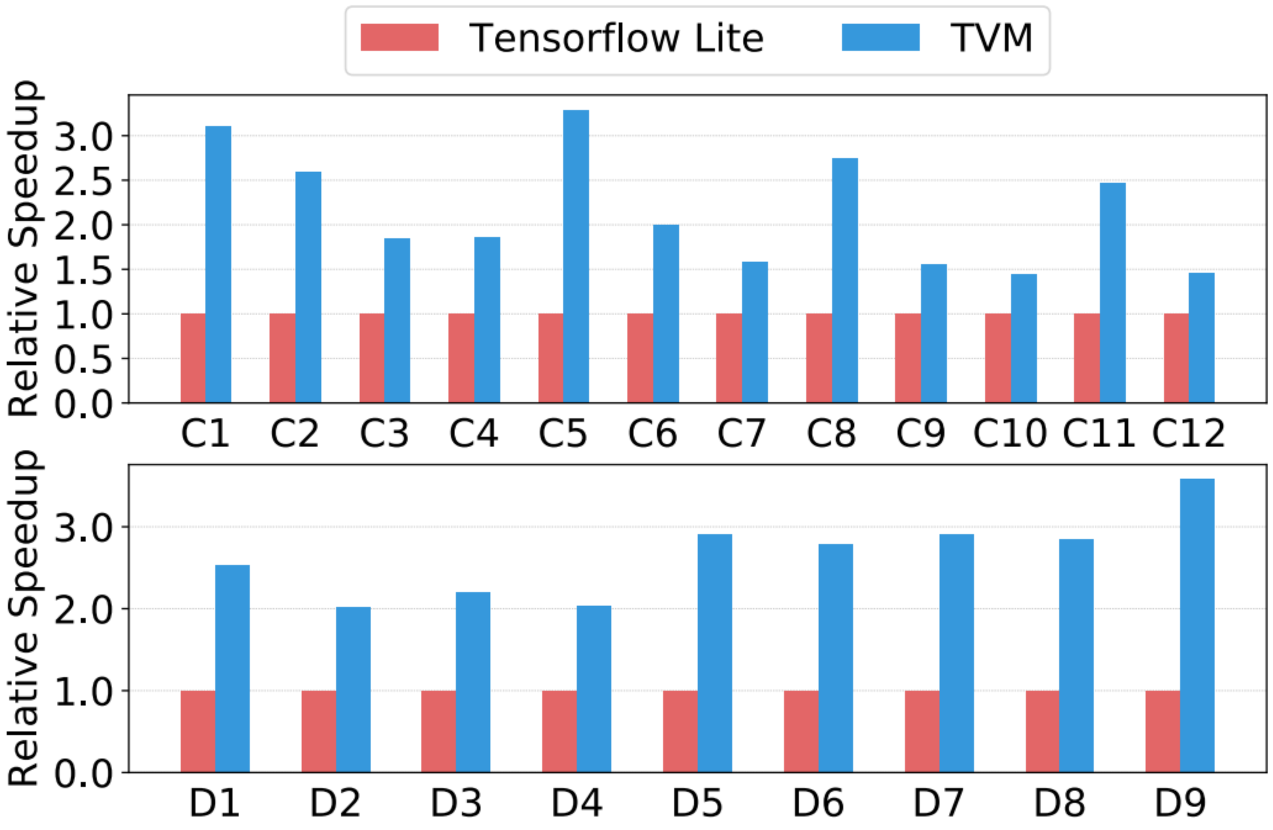


Figure 17: Relative speedup of all conv2d operators in ResNet-18 and all depthwise conv2d operators in mobilenet. Tested on ARM A53. See Table 2 for the configurations of these operators.

Pros and Cons

- Comprehensive, well written paper.
- Evaluation shows that TVM performs very well.
- No information on compilation times of models?
- Does not support dynamic input shapes.
- (Only for inference, not training).

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

Questions and discussion...

Chen, T., Moreau, T., Jiang, Z., Zheng, L., Yan, E., Shen, H., ... & Krishnamurthy, A. (2018). {TVM}: An automated {End-to-End} optimizing compiler for deep learning. In 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18) (pp. 578-594).