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

R244: Large-Scale Data Processing and Optimisation

Kian Cross

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.

```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))
```
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):
        Cl[y][x] = 0
    for ko in range(128):
        for yi in range(8):
            for xi in range(8):
                Cl[y][x] += A[ko][y] * B[ko][x]}

**Cache Data on Accelerator Special Buffer**

CL = s.cache_write(C, vdata.accc_buffer)
AL = s.cache_read(A, vdata.inp_buffer)
# additional schedule steps omitted

**Map to Accelerator Tensor Instructions**

s[CL].tensorize(yi, vdata.gemm8x8)

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

![Graph](image)

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...