Analyzing and Enhancing Quantization in TVM

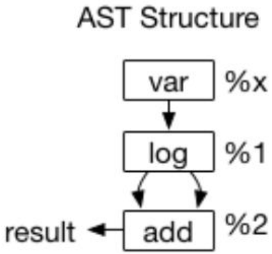
06/13/2023

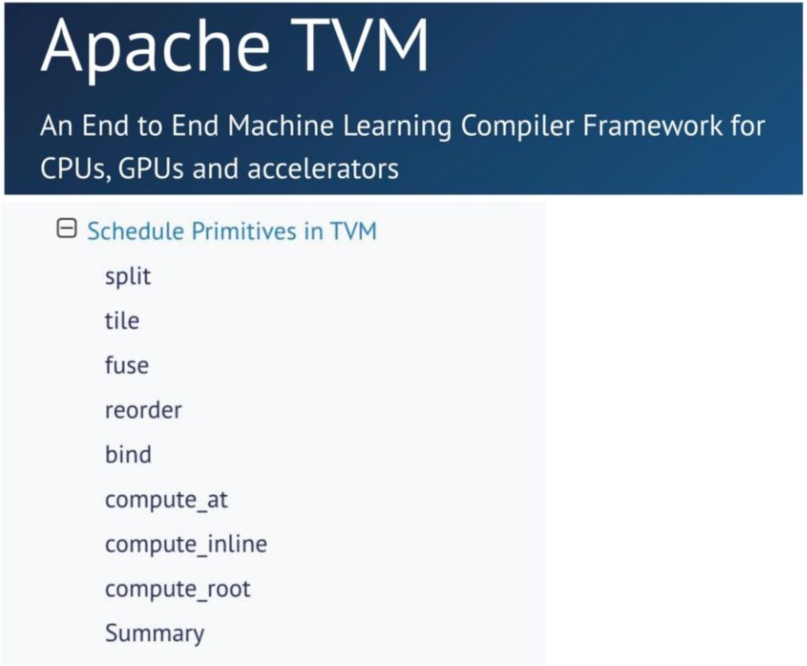
Mingfei Guo

TVM: ML Compiler Inspired by Halide

Text Form

```
fn (%x) {
  %1 = log(%x)
  %2 = add(%1, %1)
  %2
}
```



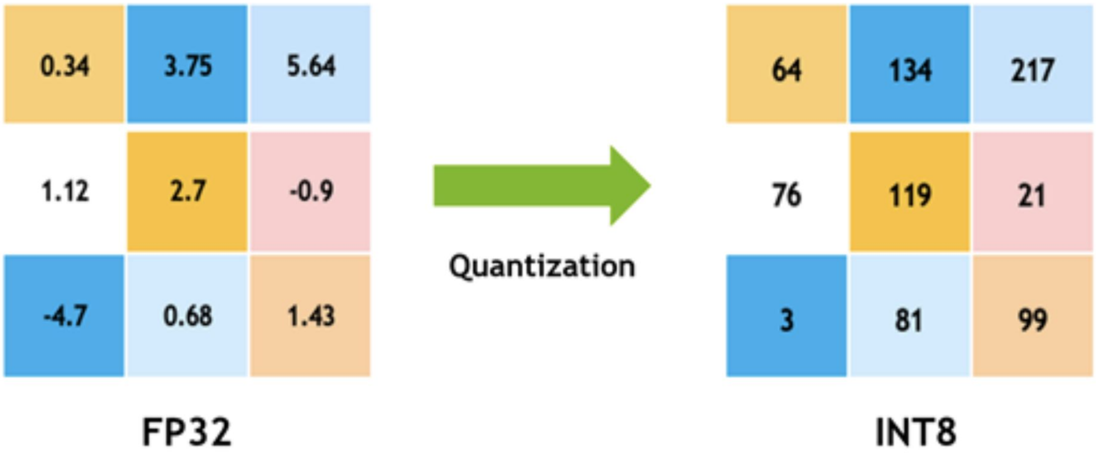


Two levels of optimization:

- graph level: dataflow rewriting and memory management, e.g. inlining, layout transformation
- tensor level: new schedule primitives

Quantization Benefits: Computation and Memory

- Quantization brings two advantages: less memory/bw usage, faster inference. Ideally 2-4x improvement from fp32 to int8.
- For our ResNet-18 task: Small batch size is computation bound, and large batch size is memory bound.



Naive TVM Quantization: 2x Worse Than FP32

- Our first fix, graph level optimization: use static graph executor (static model like ResNet) instead of the default vm executor (enables dynamic model like RNN). Make the result reasonable.
- This int8 quantization benefit from CPU lowbit inference.

Table 1: **ResNet-18 inference result.** PyTorch and TVM represent the original model and the TVM-compiled fp32 model. TVM-Quant refers to the original TVM-compiled int8 quantized model, while TVM-Quant-SG represents our fixed version of the TVM-compiled int8 quantized model.

Framework	Layout	Schedule	Precision	Time (ms)	Improvement
PyTorch	NCHW	-	fp32	69.26	-
TVM	NCHW	nchw_spatial_pack	fp32	13.29	100%
TVM-Quant	NCHW	nchw_spatial_pack	int8	29.19	45.52%
TVM-Quant-Graph	NCHW	nchw_spatial_pack	int8	8.27	160.70%

Computation Bound: Decided by Schedule Quality

- Ideal: Orthogonal optimization, significant improvement.
- Reality: int8 quantization invokes a different strategy in TVM, doesn't make sense to compare between strategies

Table 2: **ResNet-18 inference with batch size 1 under the fixed framework TVM-Quant-Graph.** The optimizations are not orthogonal, as different settings would map to different schedules, and these schedules are optimized to varying degrees. The improvement for computation-bound tasks depends on the quality of the setup and the schedule.

Layout	Schedule	Precision	Time (ms)	Ideal Speedup
NCHW	spatial_pack	fp32	13.29	16x
	spatial_pack	int8	8.27	16x
	simd	int8	11.36	16x
NHWC	spatial_pack	fp32	35.15	4x
	quantized_interleaved	int8	12.09	16x

Memory Bound: Benefit from Low Bandwidth

- Consistent memory: int8 in cpu, fp32 in memory, which is important for making use of registers and maintaining accuracy

Table 3: **ResNet-18 inference with batch size 64 and 256 under the best layout and schedule setup.** Here, the improvement refers to the gain achieved by using int8 precision compared to fp32 precision when all other settings remain the same. When the batch size is relatively large, the benefits of using less memory bandwidth become more apparent.

Batch Size	Memory (MiB)	Precision	Time (ms)	Improvement
1	5279	fp32	13.29	100%
	5331	int8	8.27	160.70%
64	5922	fp32	19.65	100%
	6009	int8	11.99	163.88%
256	9643	fp32	22.15	$\boxed{100\%}$
	10061	int8	11.36	$\boxed{194.98\%}$

Thanks!