Deep Learning Compiler

Yida Wang and Zhi Chen
AWS AI
A reduced snapshot of AWS AI

**AI SERVICES**
- Amazon Rekognition
- Amazon Polly
- Amazon Transcribe
- Amazon Translate
- Amazon Comprehend

**AMAZON SAGEMAKER**
- ML Marketplace
- Model training
- Model tuning
- SageMaker Autopilot
- SageMaker Neo

**ML FRAMEWORKS & INFRASTRUCTURE**
- GPUs and CPUs
- Elastic Inference
- Inferentia
- FPGA
Deep learning compiler projects at AWS AI

- Benchmarking
- Accelerator
- Runtime Integration
- Dynamic models
- Auto-tuning
- Performance optimization
- Operator coverage
- Scheduling management
- Sparsity
- Graph Optimization
- Training
- Amazon SageMaker Neo
- Community service
- Quantization
- Tutorial

© 2019, Amazon Web Services, Inc. or its Affiliates. All rights reserved.
QNN Dialect

-- Animesh Jain
How to consume a pre-quantized model via TVM?

Option 1 – Completely add new ops from scratch

• New Relay passes and TVM schedules required
  • AlterOpLayout, Graph Fusion etc require work/operator
• No reuse of existing Relay and TVM infrastructure

Option 2 – Lower to a sequence of existing Relay operators

• We introduced a new Relay dialect – QNN to encapsulate this work
• Complete reuse of Relay pass infrastructure
• Possible reuse of TVM schedules (only to some extent)
QNN Dialect

• Design operators that satisfy many framework operators
  • qnn.quantize, qnn.dequantize, qnn.requantize
  • qnn.conv2d, qnn.dense
  • qnn.concatenate
  • qnn.add, qnn.mul

• QNN operators will be lowered to Relay operators

• QNN Optimization passes
  • Some optimizations are easier at QNN level
  • Intel x86 VNNI requires conv input dtypes to uint8 x int8
Lowering of Qnn.Quantize

```plaintext
fn (%input_data: Tensor[(2, 5), float32]) {
  qnn.quantize(%input_data, out_dtype="uint8", output_zero_point=127, output_scale=0.5f)
}

def @main(%input_data: Tensor[(2, 5), float32]) -> Tensor[(2, 5), uint8] {
  %0 = divide(%input_data, 0.5f /* ty=float32 */) /* ty=Tensor[(2, 5), float32] */;
  %1 = round(%0) /* ty=Tensor[(2, 5), float32] */;
  %2 = cast(%1, dtype="int32") /* ty=Tensor[(2, 5), int32] */;
  %3 = add(%2, 127 /* ty=int32 */) /* ty=Tensor[(2, 5), int32] */;
  %4 = clip(%3, a_min=0f, a_max=255f) /* ty=Tensor[(2, 5), int32] */;
  cast(%4, dtype="uint8") /* ty=Tensor[(2, 5), uint8] */
}```
Lowering of Qnn.Conv2D

For zero-centered zero point, the lowering will have just `nn.conv2d`

```python
fn (%data: Tensor[(1, 3, 2, 3), uint8], %weight: Tensor[(3, 3, 2, 2), uint8]) {
    qnn.conv2d(%data, %weight, ... , out_dtype="int32", input_zero_point=1, kernel_zero_point=1)}
```

```python
def @main(%data: Tensor[(1, 3, 2, 3), uint8], %weight: Tensor[(3, 3, 2, 2), uint8]) -> Tensor[(1, 3, 1, 2), int32] {
    %0 = nn.conv2d(%data, %weight, ... , out_dtype="int32") /* ty=Tensor[(1, 3, 1, 2), int32] */;
    %1 = cast(%data, dtype="int32") /* ty=Tensor[(1, 3, 2, 3), int32] */;
    %2 = multiply(%1, 4 /* ty=int32 */) /* ty=Tensor[(1, 3, 2, 3), int32] */;
    %3 = nn.avg_pool2d(%2, pool_size=[2, 2]) /* ty=Tensor[(1, 3, 1, 2), int32] */;
    %4 = sum(%3, axis=[1], keepdims=True) /* ty=Tensor[(1, 1, 1, 2), int32] */;
    %5 = multiply(1 /* ty=int32 */, %4) /* ty=Tensor[(1, 1, 1, 2), int32] */;
    %6 = subtract(%0, %5) /* ty=Tensor[(1, 3, 1, 2), int32] */;
    %7 = cast(%weight, dtype="int32") /* ty=Tensor[(3, 3, 2, 2), int32] */;
    %8 = sum(%7, axis=[1, 2, 3]) /* ty=Tensor[(3), int32] */;
    %9 = reshape(%8, newshape=[1, 3, 1, 1]) /* ty=Tensor[(1, 3, 1, 1), int32] */;
    %10 = multiply(1 /* ty=int32 */, %9) /* ty=Tensor[(1, 3, 1, 1), int32] */;
    %11 = subtract(12 /* ty=int32 */, %10) /* ty=Tensor[(1, 3, 1, 1), int32] */;
    add(%6, %11) /* ty=Tensor[(1, 3, 1, 2), int32] */
}
```

Asymmetric
Proof of Concept on TFLite models

<table>
<thead>
<tr>
<th>Unit: ms</th>
<th>Float32</th>
<th>Quantized</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inception V1</td>
<td>NA</td>
<td>2.143</td>
<td>NA</td>
</tr>
<tr>
<td>Inception V2</td>
<td>NA</td>
<td>8.919</td>
<td>NA</td>
</tr>
<tr>
<td>Inception V3</td>
<td>10.373</td>
<td>6.115</td>
<td>1.7</td>
</tr>
<tr>
<td>Inception V4</td>
<td>21.233</td>
<td>12.315</td>
<td>1.72</td>
</tr>
</tbody>
</table>

- Metric – Latency in ms for batch size = 1
- Comparable accuracies
- 1.7x speedup on Inception asymmetric quantized model
- **Symmetric** model improves the speedup to 2.8x
How Can We Make It Better?

Framework Pre-quantized Graph

- TF
- Mxnet
- QNN Dialect

... parsers using QNN Graph

QNN passes

Relay Int8 Graph

Target-independent Relay passes

- Target-dependent Relay layout opt

Target-optimized Int8 Relay Graph

- Intel x86
- ARM CPU
- Nvidia GPU
- ARM GPU

... schedules
New Amazon EC2 instances

-- Hongbin Zheng, Yizhi Liu, Haichen Shen, and many people in Annapurna Labs
Amazon EC2 Inf1 instances

- Powered by AWS Inferentia
- Low latency, 3x higher throughput, up to 40% lower cost-per-inference compared to G4
- Up to 2,000 TOPS at sub-millisecond latency
- Integrated with popular ML frameworks TensorFlow, PyTorch and MXNet
AWS Inferentia Chip

- Four NeuroCores per chip
  - customized tensor-level optimization
- Two-stage memory hierarchy: large on-chip cache and commodity DRAM
  - Proactive data movement management
- Fast chip-to-chip interconnect via specialized communication protocol
  - Model parallelism in pipeline
Amazon EC2 M6g, R6g, C6g instances

- Powered by ARM-based AWS Graviton2 processors
- 4x more compute cores, 5x faster memory, and 7x the performance of initial Graviton offering
- 40% price/performance advantage over current x86-based instances
ML inference on Graviton2

• General-purposed CPU is capable of doing machine learning/deep learning inference
  • Check out our paper *Optimizing CNN Model Inference on CPUs* at USENIX ATC ’19

• Compared to M5, M6g does faster model inference with lower price
Dive into Deep Learning Compiler

-- Mu Li and Yida Wang
A typical conversation

**Customer/user/new hire/…**

- How to use TVM to do…
- Cool, is there any tutorial?
- I failed. TVM is REALLY REALLY hard to get started/use/deploy…

**Us/and maybe you…**

- We can do the following steps…
- Yes, for this check this, for that check that…

© 2019, Amazon Web Services, Inc. or its Affiliates. All rights reserved.
Hi,
I really want to be able to help out on this project and other DL edge frameworks, but I need some guidance on where to start.
Dive into Deep Learning ([https://D2L.ai](https://D2L.ai))

- An interactive deep learning book with code, math, and discussions
- GitHub: 18,000 stars, 200 contributors
- Presented in multiple languages: Chinese, English, Japanese, Korean
- The Chinese book is the No. 1 best seller at the largest Chinese online bookstore
- The English book is available in preview
D2L adoption

- Adopted as a textbook by 40+ global universities and Amazon Machine Learning University
  - Carnegie Mellon University
  - Indian Institute of Technology Bombay
  - Massachusetts Institute of Technology
  - Peking University
  - Shanghai Jiao Tong University
  - University of California, Berkeley
  - University of Illinois at Urbana-Champaign
  - University of Science and Technology of China
  - Zhejiang University
D2L Compiler: [http://tvm.d2l.ai](http://tvm.d2l.ai)

- A systematic tutorial to the beginners who want to **USE** TVM, and more broadly, who’d like to take DLC-101
- Python notebook based, runnable on Colab
- V0.1 released, 22 sections, covering getting started and basic operator-level optimization
- Call for contributors
SageMaker Neo

-- Amazon SageMaker Neo team
SageMaker Neo: Train once, Run anywhere

SageMaker Algorithms

ONNX
mxnet
TensorFlow
PYTORCH
XGBoost
arm
cadence
intel
NVIDIA
RISC-V
XILINX

© 2019, Amazon Web Services, Inc. or its Affiliates. All rights reserved.
Amazon SageMaker Neo Pipeline

- Input models
- Compilation configuration
- Compilation
- S3 bucket
- docker

© 2019, Amazon Web Services, Inc. or its Affiliates. All rights reserved.
Integration with SageMaker NEO

Compiler

- TensorFlow Parser
- MXNet Parser
- PyTorch Parser
- ONNX Parser
- XGBoost Parser

- Relay
- TVM
  - LLVM
  - CUDA
  - OpenCL

- Contrib Compiler

Runtime

- Compiled Model
- Neo Runtime
  - Contrib Runtime
- EC2 Instance
- Device

- X86
- AARCH64
- RISC-V
- GPU
- FPGA
- ASIC
Bring Your Own Codegen to TVM

-- Zhi Chen and Cody Yu
Why?

ResNet-50

conv + max pool
conv + max pool
conv
conv
conv + max pool
  
conv
conv
conv + max pool
  
conv
conv
conv + max pool
  
conv + up sample
concatenate + conv

Detected Big Objects

Detected Medium Objects

Detected Small Objects

NMS

On CPU/GPU

On Your device

NMS

Your device does not support!
How does it look like?

```
def @main(%x, %y) {
    add(%x, %y)
}
```

```
def @main(%x, %y) {
    %0 = fn (%gcc_input0, %gcc_input1, External="gcc", FuncName="gcc_0", Primitive=1) {
        add(%gcc_input0, %gcc_input1)
    };
    %0(%x, %y);
}
```
How to Integrate?

• Codegen
  ▪ With engine DNNL, TensorRT
    ▪ Generate artifacts that can be loaded/saved through existing TVM runtime module, e.g. DSOModule
  ▪ Without engine
    ▪ Produce library wrappers that are compatible to TVM and generate DSOModule

• Runtime
  ▪ Reuse existing TVM runtime
  ▪ Custom runtime could be imported to TVM runtime
  ▪ Invoke integrated runtime directly through PackedFunc
Pass Manager

-- Zhi Chen
What does it do?

• Traditional compiler
  ▪ Make pass developers’ life easier
  ▪ Maintain pass info, i.e. pass dependencies
  ▪ Keep analysis info update to date

• Deep learning framework
  ▪ PyTorch and Keras Sequential, Gluon Block
  ▪ Allow flexible customized pipeline
# How does it work?

<table>
<thead>
<tr>
<th>Relay IR</th>
<th>Compilation and Optimization (PM)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Module Pass</td>
</tr>
</tbody>
</table>

- Apply a sequence of passes
- Help developers to customize optimization pipeline
- Fold scale axis, hardware dependent/independent passes
How to Customize Your Optimization Pipeline?

seq1 = relay.transform.Sequential([a, b, c])
seq2 = relay.transform.Sequential([d, e, f])

with build_config(opt_level=2): # hardware independent
    mod1 = seq1(mod)

with build_config(opt_level=3, disabled_pass=[e]): # hardware dependent
    mod2 = seq2(mod1)

Tutorial: https://docs.tvm.ai/tutorials/dev/relay_pass_infra.html#sphx-glr-tutorials-dev-relay-pass-infra-py
AWS in rest of today

- 12:10 Dynamic Execution and Virtual Machine
- 13:20 Dynamic Model - Graph Dispatching
- 17:30 Improving AutoTVM Efficiency by Schedule Sharing
- 17:40 Optimizing Sparse/Graph Kernels via TVM
Takeaways

• Industry needs an open standard compiler for DL
  • AWS working on the TVM stack

• We are eager to collaborate with the community
  • Talk to us, we have 10+ people here today!

• We are hiring!
  • Write to Yida Wang (wangyida@amazon.com), Zhi Chen (chzhi@amazon.com), or Vin Sharma (vinarm@amazon.com)