TVM with PaddlePaddle

Jiajun Jiang, Staff R&D at PaddlePaddle
YuGuang Deng, Staff R&D at Baidu
Yin Ma, Principal Architect at Baidu USA
SCHEDULE

01 PaddlePaddle Frontend
02 Speed up Paddle.js on web browser
03 TVM for KunlunXin Chip
Overview of PaddlePaddle

- Agile framework for Industrial-level development of deep neural networks
- Supports Ultra-Large-Scale training of deep neural networks
- High-Performance inference over ubiquitous environments
- Industry-leading and fully open sourced models and development kits
## Multi-industry Model Zoo

More than 200+ Official models, covering NLP/CV/Speech

<table>
<thead>
<tr>
<th>E-2-E Development Kits</th>
<th>Model Zoo</th>
<th>Task Level</th>
<th>Algorithm level</th>
<th>Framework and tools level</th>
<th>Industrial Deployment</th>
</tr>
</thead>
<tbody>
<tr>
<td>PaddleOCR</td>
<td>PaddleNLP</td>
<td>Object Detection</td>
<td>DenseLab-v, SiamRPN, PSGNet, U-Net, LaneNet, HRNet, CRNN, Fast-SCNN</td>
<td>Paddle Slim</td>
<td>Paddle.js</td>
</tr>
<tr>
<td>PaddleDetection</td>
<td>ERNIE</td>
<td>Lexical Analysis</td>
<td>S3D, Faster-RCNN, Mask-RCNN, RetinaNet, YOLOv3, CRNN, GCNet, Libra R-CNN, FCOS, EfficientDet, CornerNet, FPN, HRnet</td>
<td>Paddle</td>
<td>Paddle.js</td>
</tr>
<tr>
<td>PaddleSeg</td>
<td>Parakeet</td>
<td>Emotion Analysis</td>
<td>Senta, EmoText, EmotionDetection</td>
<td>Paddle</td>
<td>Paddle.js</td>
</tr>
<tr>
<td>PaddleSeg</td>
<td>PaddleRec</td>
<td>Speech Recognition</td>
<td>DeepSpeech, DeepSR</td>
<td>Paddle</td>
<td>Paddle.js</td>
</tr>
</tbody>
</table>

### More than 200+ Official models

- NLP: PLSC, Large Scale Classification
- CV: PaddleSlim
- Speech: PaddleLite, Paddle.js
import paddle
import paddle.vision.models as models

model = models.resnet50(pretrained=True)
model.eval()

# save model as static model
input_spec = paddle.static.InputSpec(dtype="float32",
                                     shape=[None, 3, 224, 224], name="image")
paddle.jit.save(model, "save_dir/model", [input_spec])

Export PaddlePaddle model for deployment

import paddle
from tvm import relay

model = paddle.jit.load("./inference/model")
mod, params = relay.frontend.from_paddle(model)

with tvm.transform.PassContext(opt_level=3):
    lib = relay.build(mod, target, params=params)

Convert to TVM Relay by PaddlePaddle frontend

For more details please refer to TVM tutorials: https://tvm.apache.org/docs/how_to/compile_models/from_paddle.html
PaddlePaddle Frontend for TVM

• Plans of PaddlePaddle Frontend
  • Support 200+ PaddlePaddle operators
  • Control Flow operators
  • Quantize Model(QAT) by PaddleSlim
02  Speed up Paddle.js on web browser

Yuguang Deng
Paddle.js Team
Architecture and Motivation

• Overview of Paddle.js
  • Paddle.js is a library for executing machine learning algorithms in JavaScript.
  • Paddle.js models run in a web browser Mini Program, Node.js environment
  • Paddle.js is part of the PaddlePaddle ecosystem, allowing more developers to migrate from JavaScript to machine learning community

• Motivation
  • Mobile devices are very fragmented and lack of standard API to access GPU and CPU directly in web browser.
  • Operators in the models are not fully optimized
  • WASM runtime is supported on TVM
  • The new RPC Session via websocket makes it possible for AutoTVM to run in the browser
Development

- Create the model and runtime through TVM, and then optimize it with AutoTVM in the browser.
- Track the running performance of TVM WASM runtime and Paddle.js respectively via profiler.
- Pick the best kernel implementations from the above. And then combine to new graph.
Performance

- Face detection model is taken as an example from online application
- The end-to-end performance speedup 10% on lower device. The replaced kernel is not optimized by manually.
- TVM can help us locate the non optimized parts and give better solutions automatically
Future work

• Online scenarios need to ensure the security of model assets, for example code confusion

• Extend the studies on other backend, like WebGPU、WASM with SIMD and muti threading APIs to fully utilize the GPU & CPU resources.
TVM for KunlunXin GP-AI Chip Family

Yin Ma
KunlunXin Compiler Team
Architecture

• Fully utilize the existing TVM framework.
  • Identical user experience coming from TVM
  • Use GraphRuntime to support static models
  • Use VM + BYOC to support dynamic models or models with control flow
  • Powerful Target specific optimizer
  • Massive XPU TOPI to call optimized device kernel library
  • Support Linux, Windows, X86, Aarch64 and other platforms

• Performance strategy
  • Map most operators to call optimized device kernel implementation.
  • Use TIR code generation to cover the long tail pattern.
  • Make all possible operators to run on device to reduce the cost of device copy

*XPU is the architecture name of one KunlunXin chip family.
Development

- **Model Importing**
  - Parser for PaddlePaddle models, provided by Paddle TVM team.
  - Improve other parsers to import more models and dynamic shape
  - C++ wrapper to enable network creation, build and run in pure C++

- **Relay Optimization**
  - Convert to a double linked graph to enable large scale and complex rule-based pattern matching, operator replacement for fusion, var-length support etc.
  - Add device specific types and passes to support quantization
  - Add operators to support model parallelism inferencing

- **Backend**
  - New codegen to generate XPU C/C++ code and drive XPU LLVM based compiler
  - New BYOC connected XPU inferencer designed to handle dynamic shape in first place

- **Runtime**
  - Automatic device memory hierarchy optimization algorithm
  - Configuration file based memory location assignment framework
  - Support for dumping values in device memory after each layers for debugging
Performance

- Well deployed already in many industry inference applications such as searching, quality inspection etc.
- Proven capability to deliver the peak chip performance for all kind of models in the real business engagement.

* Data above came from the testing with a fair setup between KunlunXin R200 accelerator and a comparable industry mainstream accelerator using our TVM based compiler in Sept, 2021
Future work

• Current limitation
  • Auto tune and auto schedule is not enabled
  • Some schedule primitives are not supported

• Production-driven future development
  • Passing compiled models via memory stream
  • User friendly compilation flow to make compiled models to run on devices with different architectures transparently
  • Improve VM framework and reduce its runtime cost like those from dramatically increased operator counts
  • Need a way to reduce cost from date structure used in runtime like TVMArgs, such as hardening the type checking
  • Define new schedule primitives to fit KunlunXin hardware better
  • Engage with community for better collaboration and upstreaming

• We are hiring
  • Email to Yin Ma (yinma@baidu.com), Beijing, Shanghai, USA
Thanks!