LPCV2021 FPGA Track Winner Solution: When industrial model toolchain meets Xilinx FPGA

Jiahao Hu, Ruihao Gong

SenseTime Research, Model Toolchain Team
Team Members

Jiahao Hu
Detection, Optimization

Pu Li
Quantization, Compile

Yongqiang Yao
Detection, Optimization

Ruihao Gong
Quantization, Optimization

Shuo Wu
Detection

Yucheng Wang
FPGA board setup

Liang Liu
FPGA board setup

Yusong Wang
Inference Optimize

Fengwei Yu
Consultant
<table>
<thead>
<tr>
<th>Part</th>
<th>Title</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Overview</td>
<td>P04-P07</td>
</tr>
<tr>
<td>2</td>
<td>Model Training</td>
<td>P08-P11</td>
</tr>
<tr>
<td>3</td>
<td>Quantization, Optimization</td>
<td>P12-P16</td>
</tr>
<tr>
<td>4</td>
<td>Result and Summarization</td>
<td>P16-P17</td>
</tr>
</tbody>
</table>
A typical pipeline for model production

Overview

1. **Training System**
   - Data
   - Label
   - PyTorch
   - Caffe
   - TensorFlow
   - GPU
   - CPU
   - TPU

2. **Deployment System**
   - hardware independent model
   - Limited training hardware

3. **Solution system**
   - GPU
   - CPU
   - NPU
   - DSP
   - image
   - Decode
   - Detect
   - Track
   - result
   - Pre-/Post-processing pipeline

4. **Hardware**
   - Cloud Server
   - Edge device
   - Mobile / IOT device

Hardware-related model
Hardware-independent model
Overview

Our toolchain system for efficient model production

United Perception training framework
Model quantization toolkit
Multi-platform model deployment system

Training
- One for all: classification, detection, segmentation, etc.
- Bag of tricks: summarize the best practices for training
- Large-scale: large-scale dataset training support
- Deployable: hardware friendly

Quantization
- SOTA algorithms: LSQ, Brecq, Qdrop, etc.
- Deployable: quantized models can be directly exported to hardware format
- Flexible: automatic quantization node insertion

Deployment
- Dozens of hardware platform support: NV GPU, DSPs, etc.
- Compiler and runtime: support mixed backend and multi-platform model conversion and inference
- Code level, operator level and network level integration

https://github.com/ModelTC/United-Perception  https://github.com/ModelTC/MQBench

Not open-sourced yet.

The toolchain system is based on powerful algorithms and numerous engineering efforts.
Comprehensive quantization algorithms: DFQ -> PTQ -> QAT

DSG CVPR2021 Oral

BRECQ ICLR2021

Block reconstruction for PTQ, 4bit close to QAT

QDrop ICLR2022 Top1%

Case 1: block 1 → block k-1 → block k
Case 2: block 1 → block k-1 → block k
Case 3: block 1 → block k-1 → block k

Randomly dropping for better PTQ, new SOTA

Calibration data is crucial

Data diversity greatly helps data-free quantization

MixMix ICCV2021

Multi-model generation is good for the fidelity.

Reproducible and deployable, bridge the hardware and algorithms, followed by Intel and Qualcomm

MQBench NeuIPS2021 benchmark track

OQAT ICCV2021

NAS: Quantization-friendly architectures

Solve the last mile problem

Calibration data is crucial

Data diversity greatly helps data-free quantization

Multi-model generation is good for the fidelity.

Reproducible and deployable, bridge the hardware and algorithms, followed by Intel and Qualcomm

NAS: Quantization-friendly architectures

Solve the last mile problem
Overview

Objective of LPCV 2021 FPGA Track Challenge

**Vision Task:** Object detection.

**Data:** Coco 2017 images data download link: [https://cocodataset.org/#download](https://cocodataset.org/#download)

**Hardware:** [Ultra96-V2](https://www.xilinx.com/products/boards-and-cards/ultra96-v2.html) + Xilinx® Deep Learning Processor Unit (DPU)

**Software:** PYNQ

**Evaluation:** $10^4 / \text{Energy} \times \text{ReLU}(m\text{MAP} - 0.2) \times \text{ReLU}(\text{fps} - 5)$. Where $m\text{MAP}$ is INT8 quantized accuracy

- Hardware: Ultra96-V2
- Inference lib: Vitis AI
- Quantization scheme: Ristretto
- Task: object detection

A scene similar to that we often face in the industry production Can be handled with our toolchain system.
LPCV FPGA Track Winner Solution

Overall pipeline

- Training YOLOX-FPGA with UP
- Quantizing the model with MQBench
- Compile to xmodel
- Evaluate on the board
- Optimize the bottleneck
YOLOX-FPGA: Hardware-software-algorithm co-design

Existing YOLOX models can not directly deploy on the FPGA board => hardware friendly design

**Width Selection**

Best setting: width=0.375, depth=0.33

<table>
<thead>
<tr>
<th>Model</th>
<th>Width</th>
<th>Image size</th>
<th>FP32</th>
<th>INT8</th>
<th>HW</th>
<th>Latency (ms)</th>
<th>Total Energy (J)</th>
<th>FPS</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>YOLOX-FPGA</td>
<td>0.375</td>
<td>416</td>
<td>32.1</td>
<td>30.2</td>
<td>30.0</td>
<td>38.906</td>
<td>635.876</td>
<td>25.702</td>
<td>34.510</td>
</tr>
<tr>
<td>YOLOX-FPGA-nospp</td>
<td>0.3125</td>
<td>416</td>
<td>26.4</td>
<td>24.9</td>
<td>24.8</td>
<td>32.232</td>
<td>548.888</td>
<td>31.025</td>
<td>22.759</td>
</tr>
<tr>
<td>YOLOX-FPGA</td>
<td>0.3125</td>
<td>416</td>
<td>28.1</td>
<td>20.4</td>
<td>No</td>
<td></td>
<td>No</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>YOLOX-FPGA</td>
<td>0.25</td>
<td>416</td>
<td>22.5</td>
<td>20.4</td>
<td>20.3</td>
<td>27.575</td>
<td>No</td>
<td>36.265</td>
<td>None</td>
</tr>
</tbody>
</table>

**OP adaptation**

Considering the supported ops by Xilinx Ultra96 Vitis AI:
1. SiLU → ReLU;
2. SPP MaxPool2d (ceil_mode=False) → dilated convs;
3. Focus → three Conv2d (kernel_size=3);
YOLOX-FPGA: Hardware-software-algorithm co-design

Experimenting different input shapes to achieve best accuracy-latency trade-off.

Best setting: gray input, input size = 320

<table>
<thead>
<tr>
<th>Model</th>
<th>image size</th>
<th>FP32</th>
<th>INT8</th>
<th>HW</th>
<th>Latency(ms)</th>
<th>Total Energy(J)</th>
<th>FPS</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>YOLOX-FPGA</td>
<td>288</td>
<td>26.9</td>
<td>24.2</td>
<td>24.6</td>
<td>21.817</td>
<td>378.108</td>
<td>45.837</td>
<td>49.681</td>
</tr>
<tr>
<td>YOLOX-FPGA</td>
<td>320</td>
<td>29.05</td>
<td>27.2</td>
<td>27.4</td>
<td>24.557</td>
<td>416.239</td>
<td>40.723</td>
<td>63.509</td>
</tr>
<tr>
<td>YOLOX-FPGA</td>
<td>352</td>
<td>31.1</td>
<td>29.3</td>
<td>29.3</td>
<td>28.835</td>
<td>480.148</td>
<td>34.658</td>
<td>57.444</td>
</tr>
<tr>
<td>YOLOX-FPGA</td>
<td>416</td>
<td>34.1</td>
<td>32.2</td>
<td>32.2</td>
<td>38.838</td>
<td>635.785</td>
<td>25.748</td>
<td>49.407</td>
</tr>
</tbody>
</table>
Improving accuracy with training techniques in UP

**Pre-training**

1. Select images in Object 365 with the same classes as MSCOCO according to Unidet class map and pretrained YOLOX-FPGA.
2. Finetune YOLOX-FPGA with 1/10 lr of pretraining.

**Knowledge Distillation**

1. Train YOLOX-medium (4 times YOLOX-FPGA parameters) on MSCOCO.
2. Distill features of neck outputs with AT-loss.

<table>
<thead>
<tr>
<th>Model</th>
<th>Pre-training</th>
<th>Knowledge Distillation</th>
<th>AP(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>YOLOX-FPGA</td>
<td></td>
<td></td>
<td>25.69</td>
</tr>
<tr>
<td>YOLOX-FPGA</td>
<td>√</td>
<td></td>
<td>27.77 (+2.08)</td>
</tr>
<tr>
<td>YOLOX-FPGA</td>
<td>√</td>
<td>√</td>
<td>29.04 (+1.27)</td>
</tr>
</tbody>
</table>

Comparison of different improvements AP on COCO 2017 val. Test size is 320
Post-processing optimization

- Efficient NMS implementation: using python => using cython
- Post-processing pipeline optimization
### Post-processing optimization

<table>
<thead>
<tr>
<th>Improvement</th>
<th>NMS</th>
<th>Few sigmoid</th>
<th>Few offset decoder</th>
<th>Time (ms/img)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Post-process</td>
<td></td>
<td></td>
<td></td>
<td>31.6968</td>
</tr>
<tr>
<td>Post-process</td>
<td>✓</td>
<td></td>
<td></td>
<td>17.7001</td>
</tr>
<tr>
<td>Post-process</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>10.699</td>
</tr>
<tr>
<td>Post-process</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>8.5264</td>
</tr>
</tbody>
</table>

Comparison of different improvements cost time (COCO 2017 test-dev). The NMS is efficient cython NMS. Few sigmoid and decode are generating scores and bounding boxes after thresh filter.
LPCV FPGA Track Winner Solution

Model Quantization => No accuracy degradation with the powerful algorithm

• MQBench: Vitis AI Backend

We provide the example to deploy the quantized EOD model to Vitis, which is winner solution for the Low Power Computer Vision Challenge 2021 (LPCV2021).

• First quantize model in EOD.

```
1  python -m eod train -e --config configs/det/yolox/yolox_fpga_quant_vitis.yaml --nm 1 --ng 1 --launch pytorch 2>&1 | tee log_qat_mq
```

• Second export the quantized model to ONNX [mqbench_qmodel.onnx] and [mqbench_qmodel_deploy_model.onnx].

```
1  python -m eod quant_deploy --config configs/det/yolox/yolox_fpga_quant_vitis.yaml --ckpt [model_save_path] --input_shape [input_shape]
```

• Third build Docker from Dockerfile, convert ONNX to xmodel [mqbench_qmodel_deploy_model.onnx_int.xmodel].

```
1  python -m mq.dep.convert_xir -Q [mqbench_qmodel.onnx] -C [mqbench_qmodel_deploy_model.onnx] -N [model_name]
```

• Fourth compile xmodel to deployable model [mqbench_qmodel.xmodel].

```
1  vai_c_xir -x [mqbench_qmodel_deploy_model.onnx_int.xmodel] -a [new_arch.json] -o [output_path] -n [model_name]
```

Heterogeneous computing optimization

Image reading, pre-processing, inference and post-processing are parallelly executed by 2 threads, respectively.

The latency is dominated by inference.

Read-Queue1 → Pre-Queue1 → Post-Queue1
Read-Queue2 → Pre-Queue2 → Post-Queue2

Read Thread 1: images
Read Thread 2: 22ms
Pre-process Thread 1: 10ms
Inference Thread 1: 25ms
Post-process Thread 1: 14ms

Results

DPU+ARM
Result: winner with the lowest energy, highest accuracy, and smallest latency

<table>
<thead>
<tr>
<th>Rank</th>
<th>Team</th>
<th>Date and Time</th>
<th>AP</th>
<th>Latency (ms)</th>
<th>Energy</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st</td>
<td>spring</td>
<td>2021-08-31 23:57:54</td>
<td>0.274</td>
<td>26.625</td>
<td>200.283</td>
<td>120.296</td>
</tr>
<tr>
<td>2nd</td>
<td>spring</td>
<td>2021-08-30 23:58:07</td>
<td>0.262</td>
<td>27.064</td>
<td>201.394</td>
<td>98.357</td>
</tr>
<tr>
<td>3rd</td>
<td>MIT HANLAB</td>
<td>2021-08-31 09:26:22</td>
<td>0.241</td>
<td>34.849</td>
<td>251.814</td>
<td>38.58</td>
</tr>
<tr>
<td>4th</td>
<td>spring</td>
<td>2021-08-28 23:49:00</td>
<td>0.233</td>
<td>32.757</td>
<td>227.456</td>
<td>37.036</td>
</tr>
<tr>
<td>5th</td>
<td>MIT HANLAB</td>
<td>2021-08-30 07:26:21</td>
<td>0.237</td>
<td>34.305</td>
<td>257.28</td>
<td>34.731</td>
</tr>
<tr>
<td>6th</td>
<td>spring</td>
<td>2021-08-29 23:38:21</td>
<td>0.233</td>
<td>31.425</td>
<td>258.196</td>
<td>34.28</td>
</tr>
<tr>
<td>7th</td>
<td>spring</td>
<td>2021-08-28 00:00:07</td>
<td>0.254</td>
<td>42.057</td>
<td>311.0</td>
<td>32.603</td>
</tr>
<tr>
<td>8th</td>
<td>NYCity</td>
<td>2021-08-31 14:01:55</td>
<td>0.252</td>
<td>46.289</td>
<td>319.332</td>
<td>27.036</td>
</tr>
</tbody>
</table>
All codes for reproducing our winner solution are open-sourced.

Welcome to star and have a try on our toolchain.


**Open-sourced inference code:** [https://github.com/ModelTC/LPCV2021_Winner_Solution/](https://github.com/ModelTC/LPCV2021_Winner_Solution/)

United Perception

Model Quantization Benchmark

link: [https://github.com/ModelTC/United-Perception](https://github.com/ModelTC/United-Perception)

link: [https://github.com/ModelTC/MQBench](https://github.com/ModelTC/MQBench)
Thanks for Listening!

Q&A